Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Sistema de visión estéreo aplicado a la detección y seguimiento de objetos en conducción autónoma

Autor: Miguel Antunes García

Tutor: Luis Miguel Bergasa Pascual y Carlos Gómez Huélamo

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Sistema de visión estéreo aplicado a la detección y seguimiento de objetos en conducción autónoma

Autor: Miguel Antunes García

Director: Luis Miguel Bergasa Pascual y Carlos Gómez Huélamo

Tribunal:

Presidente: Miguel González Herráez

Vocal 1º: Ignacio Bravo Muñoz

Vocal 2º: Luis Miguel Bergasa Pascual

Calificación: .....

Fecha: .....

"Una máquina puede hacer el trabajo de 50 hombres ordinarios, pero no existe ninguna que pueda hacer el trabajo de un hombre extraordinario."

Elbert Hubbard

Agradecimientos

Mis últimos cuatro años dedicados a los estudios universitarios en Electrónica de Comunicaciones culminan con la entrega de este trabajo.

En primer lugar me gustaría agradecer a la Universidad de Alcalá por permitirme evolucionar como persona durante este tiempo, conociendo y aprendiendo de profesores y compañeros.

Agradecer a mi tutor Luis Miguel Bergasa que me acogiera en el grupo Robesafe hace unos años y me haya guiado y ayudado en mi travesía hasta este momento.

A todos mis compañeros del grupo Robesafe y especialmente a mi cotutor Carlos Gómez y a Javier de la Peña, con los que he trabajado codo con codo estos útlimos meses.

A mi familia por apoyarme y permitirme llegar a ser la persona que soy actualmente.

Por último a todos mis amigos con los que he compartido multitud de momentos y me han ayudado a sobrellevar la situación de pandemia este último año y medio.

Resumen

El trabajo realizado busca la implementación de un sistema basado en Deep Learning para la detección tridimensional de elementos del entorno de un vehículo autónomo utilizando únicamente información de cámara.

En primer lugar se ha realizado un estudio de las técnicas de detección que se encuentran en el estado del arte, de tal forma que se elige como base una técnica que se adapte a nuestras necesidades. Una vez se ha elegido el método a utilizar, se han realizado pruebas de validación de resultados trabajando sobre el dataset de detección Kitti, obteniendo métricas oficiales del benchmark del dataset y calculando varias métricas propias para completar el estudio.

Una vez realizados los ajustes necesarios, el código se ha integrado en el Docker del proyecto Techs4AgeCar para que funcione sobre ROS. De esta manera el sistema puede funcionar sobre el simulador Carla y además se comunica con la cámara y resto del proyecto si se trabaja sobre el vehículo real. Aprovechando que paralelamente se han integrado sistemas de detección basados en otros sensores también se ha realizado una late fusion.

Por último, se ha colaborado con varios compañeros del grupo de investigación Robesafe para crear un apartado de evaluación de tareas de percepción dentro de Ad-Devkit, construyendo así una herramienta que permite obtener métricas de los resultados sobre el simulador Carla.

Palabras clave: Visión artificial, Deep Learning, Estimación 3D, Fusión sensorial, Autonomous Driving Benchmark.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Lista de acrónimos

Ad-Devkit	Autonomous Driving Benchmark Development Kit.
AOS	Average Orientation Similarity.
AP	Average Precision.
API	Application Programming Interface.
BEV	Bird's Eye View.
CNN	Convolutional Neural Network.
CVPR	Conference on Computer Vision and Pattern Recognition.
E-YOLO	Expandable YOLO.
FPS	Fotogramas por segundo.
GPU	Graphics Processing Unit.
IoU	Intersection over Union.
KIT	Karlsruhe Institute of Technology.
RCNN	Region Based Convolutional Neural Networks.
ROS	Robot Operating System.
RPN	Region Proposal Network.
SVM	Support Vector Machine.
T4AC	Techs4AgeCar.
UAH	Universidad de Alcalá.
YOLO	You Only Look Once.

Capítulo 1

Introducción

1.1 Concepto de visión artificial y Deep Learning

Visión artificial o computer vision es una rama de la inteligencia artificial que busca que las máquinas y robots puedan comprender el entorno que los rodea. Existen numerosos métodos que procesan la información procedente de diferentes sensores para sintetizar información útil para el propio sistema inteligente. Estos métodos se pueden agrupar en dos grandes categorías: los algoritmos de procesamiento de información clásicos basados en Machine Learning como Support Vector Machine o K-means, y los sistemas basados en Deep Learning, los cuales han conseguido mucha importancia gracias a los avances tecnológicos de los últimos años.

Deep Learning o aprendizaje profundo es una parte de la disciplina de Machine Learning que busca enseñar a las máquinas a realizar múltiples tareas utilizando las denominadas redes neuronales. Estas redes son modelos que intentan imitar el funcionamiento del cerebro humano (figura 1.1), y que interconectan nodos o neuronas las cuales realizan diversas funciones matemáticas. Estas redes pueden servir para realizar múltiples y diversas tareas como detección de objetos en imágenes o reconocimiento de voz. Para que estos sistemas funcionen correctamente deben ser entrenados con datos similares a con los que va a trabajar, de esta manera las neuronas modifican sus pesos para adaptarse al comportamiento deseado.

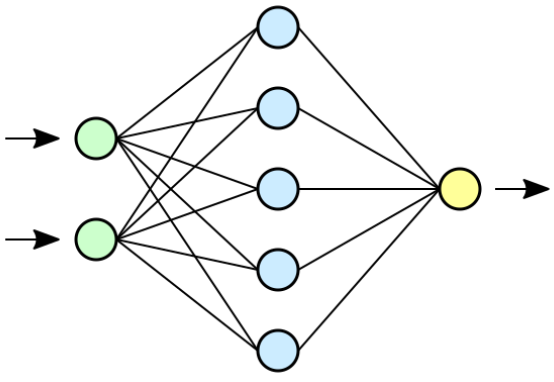


Figura 1.1: Red Neuronal simple

1.2 Cámara, LiDAR y radar

La cámara fotográfica permite capturar y almacenar una representación visual de un entorno, convirtiendo la información tridimensional en 2D. Actualmente es uno de los sensores mas extendidos en todos los ámbitos, y sobre los que mas se ha trabajado en visión artificial. Las principales ventajas de la cámara son la posibilidad de adquirir información RGB y la alta resolución que puede llegar a tener. Al realizar una transformación a un plano bidimensional se pierde por completo la información de profundidad de la escena. Este problema puede llegar a compensarse en parte utilizando cámaras estéreo (con dos objetivos), sin embargo la información de profundidad no llega a tener la precisión que aportan otros sensores.

Por otra parte, el LiDAR y radar utilizan ondas electromagnéticas para realizar la detección de elementos del entorno. En ambos casos se mide el tiempo de retorno del rayo para conocer así la presencia y distancia de los objetos. Si bien al utilizar sensores de este tipo se pierde por completo la información visual (RGB), los datos tridimensionales que se obtienen tienen errores mínimos y permiten conocer las formas de los objetos de una forma mucho mas precisa. La principal ventaja del radar es que puede detectar objetos correctamente incluso con clima desfavorable como lluvia o niebla, tarea que ni cámara ni LiDAR pueden realizar satisfactoriamente, sin embargo, pierde mucha resolución con la distancia. Por otra parte, el LiDAR consigue detectar de manera precisa también en largas distancias, sin embargo, cuenta con menos resolución que la cámara y pierde precisión con clima adverso.

En un sistema complejo sería idóneo realizar fusión de las detecciones de los tres sensores para que las fortalezas de unos puedan compensar los fallos del resto.

1.3 El proyecto Techs4AgeCar

El grupo de investigación RobeSafe se encuentra actualmente trabajando en el desarrollo de un sistema de conducción autónoma bajo el proyecto Techs4AgeCar, continuando el trabajo de un proyecto anterior.

El proyecto se puede dividir en dos secciones fundamentales: los componentes físicos como el vehículo y el resto de elementos hardware, y el software del proyecto.

Figura 1.2: Coche del grupo RobeSafe

En la figura 1.2 se puede observar el vehículo del grupo, al que se le acoplan una serie de sensores para realizar la percepción:

1. Cámara estéreo Zed: proporciona vídeo por cada una de sus lentes según las características especificadas en la tabla 1.1 y cuenta con las dimensiones mostradas en la figura 1.3.

Video mode	FPS	Resolución (sumando ambas lentes)
$2{,}2K$	15	$4416x1242$
$1080p$	30	$3840x1080$
$720p$	60	$2560x720$
WVGA	100	$1344x376$

Tabla 1.1: Modos de vídeo de la cámara Zed

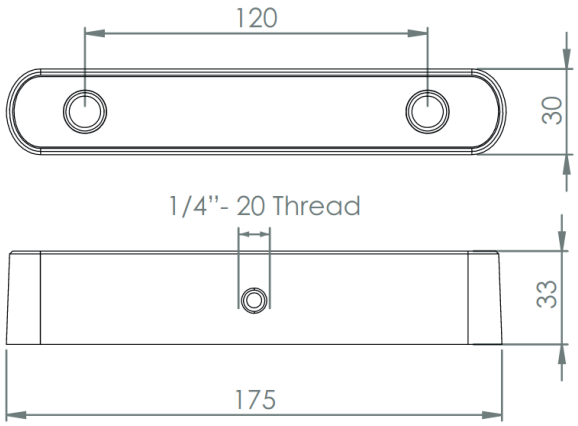


Figura 1.3: Dimensiones en mm de la cámara estéreo ZED

2. LiDAR Velodyne VLP-16: cuenta con 16 haces y con 360 grados de cobertura.

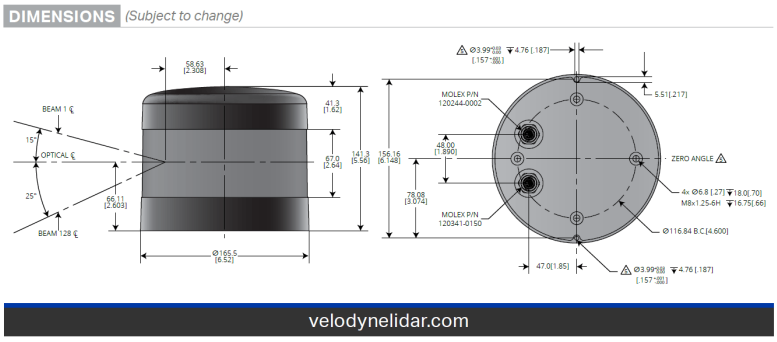


Figura 1.4: Dimensiones del LiDAR Velodyne VLP16

Respecto al software, el proyecto se encuentra por completo dentro de un contenedor de Docker y organizado en diferentes capas que agrupan sistemas con un mismo fin, por ejemplo, la capa de percepción agrupa todos los elementos que utilicen información de los sensores para analizar el entorno. La comunicación entre todos los elementos software y hardware se realiza mediante ROS (figura 1.5).

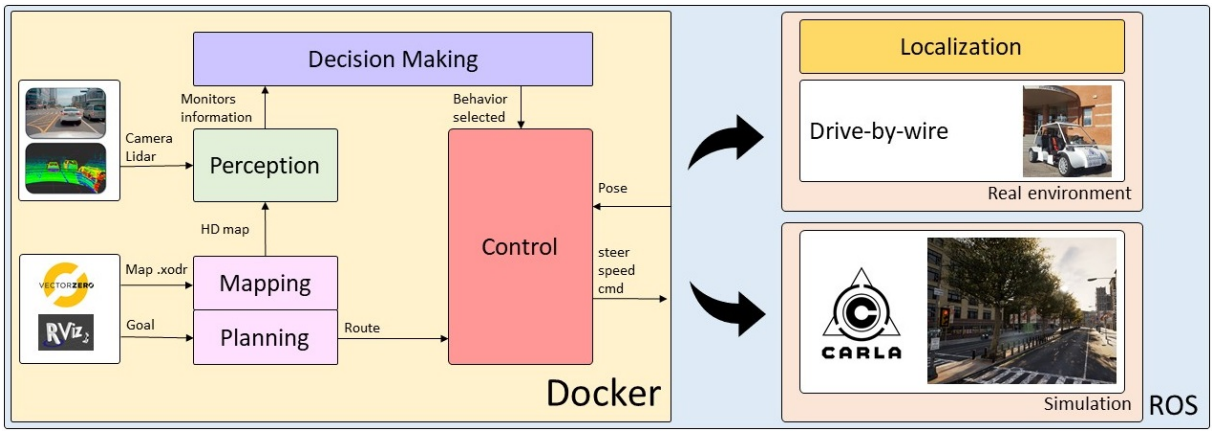


Figura 1.5: Esquema general del proyecto Techs4AgeCar

1.4 Visión general del sistema implementado

Para que la capa de percepción del proyecto T4AC adquiera robustez y proporcione unos resultados mas ajustados a la realidad, se han realizado varios trabajos en paralelo durante el curso 20-21, abarcando varios sensores presentes en el vehículo del grupo como cámara o LiDAR.

Como se va a explicar en capítulos sucesivos, en este trabajo se ha implementado un sistema capaz de generar detecciones tridimensionales de objetos a partir únicamente de la información 2D procedente de cámara. El esquema general del sistema y del sistema desglosado se muestra en las figuras 1.6 y 1.7 respectivamente.

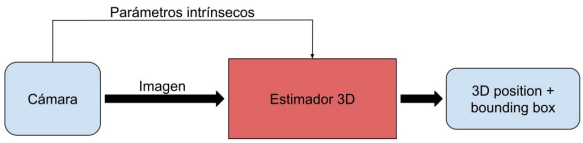


Figura 1.6: Esquema general del sistema

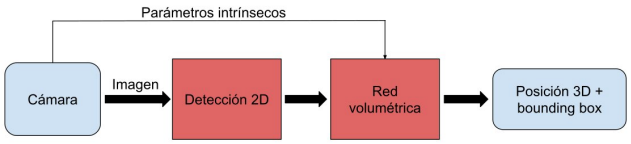


Figura 1.7: Esquema desglosado del sistema estimador 3D

Capítulo 2

Herramientas software

En este capítulo se van a explicar las herramientas software mas importantes sobre las que se va a desarrollar el trabajo, las cuales son el simulador Carla [1], ROS y Docker. El resto de software y herramientas se especificarán en el apéndice B.

2.1 Simulador Carla

El simulador Carla [1] se ha convertido en una de las mejores herramientas para el desarrollo y pruebas de sistemas de conducción autónoma. Proporciona diversos recursos open source con los que trabajar sobre simulación.

La principal ventaja del simulador es su flexibilidad a la hora de realizar configuraciones para simular los sistemas reales, permitiendo configurar las especificaciones de los diferentes sensores o controlar el entorno de simulación a través de su API.

El ROS-Bridge que incluye el simulador permite utilizar paquetes de ROS de forma mas sencilla e incluso facilitar el paso entre real y simulación de sistemas que se basen en este framework.

Figura 2.1: Capturas de Carla

2.1.1 Unreal Engine

Unreal Engine es un motor gráfico creado por Epic Games y utilizado por primera vez en 1998. Desde ese momento la compañía ha ido realizando diversas actualizaciones a lo largo de los años, mejorándolo y añadiendo características que lo han llevado a ser considerado uno de los mejores motores gráficos actualmente. En este momento, el motor se encuentra en su versión 4, habiendo publicado una beta abierta de la versión 5, la cual consigue un nivel de realismo muy elevado.

Figura 2.2: Capturas de la demo técnica de Unreal Engine 5

Carla se ejecuta sobre la versión 4 de Unreal Engine, lo cual permite a los desarrolladores utilizar las potentes herramientas que les proporciona este motor, tanto a nivel gráfico para que las simulaciones se parezcan lo mas posible a un entorno real, como a nivel de físicas con Physics o la creación y control de actores y elementos del entorno.

2.1.2 Estructura

En líneas generales el simulador consta de un mundo, unos actores, un entorno o mapa y los diversos sensores que capturan la información. Toda la información referente a actores, simulación o sensores se integra en topics de ROS a través del ROS-Bridge, por lo que es posible utilizar esta vía para recibir o enviar información a elementos del simulador. Para representar esta información de una forma mas visual se pueden utilizar programas como Rviz.

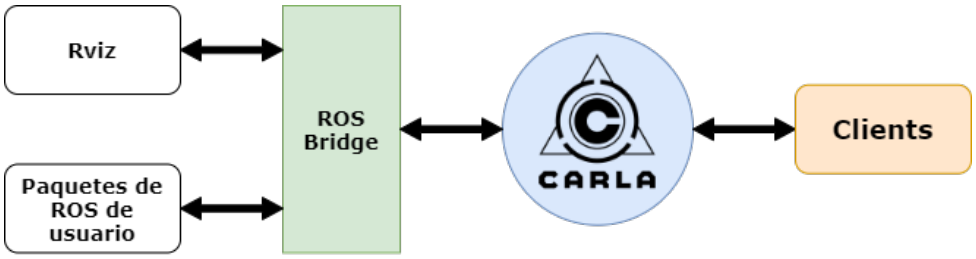


Figura 2.3: Estructura general de Carla

2.1.2.1 Mundo y clientes

El mundo es la entidad que representa la propia simulación. Conteniene la mayor parte de la información, configuración y métodos para modificar la simulación, entre los que se encuentran:

1. Información sobre los actores en la simulación y del espectador.
2. Información sobre el mapa cargado en simulación. Importante no confundir mapa con el mundo.
3. Configuración de la simulación.
4. Métodos para generar actores.

Por otra parte, los clientes son entidades separadas entre ellas y del mundo, que operan a través de scripts para conectarse al servidor (TCP-IP) y solicitar información o cambios en la simulación.

Esta relación mundo-clientes es la base de Carla. Una vez tenemos cargada la simulación, el usuario puede invocar a los clientes, que se encargan de solicitar al mundo realizar tareas como la generación de actores o cambio de mapa.

2.1.2.2 Actores

Los actores son todos los elementos presentes en la simulación que no forman parte del propio mapa. Normalmente suelen tener ciertos roles dentro del entorno virtual, suelen ser elementos dinámicos o que influyen en el comportamiento de otros actores. Los mas comunes son peatones, coches, semáforos, señales de tráfico e incluso los sensores.

2.1.2.3 Mapas y navegación

El mapa representa el propio entorno de la simulación. Por defecto Carla ofrece varios mapas propios denominados Town. Además de la información gráfica también se almacena información referente a las carreteras siguiendo el estándar OpenDrive [2].

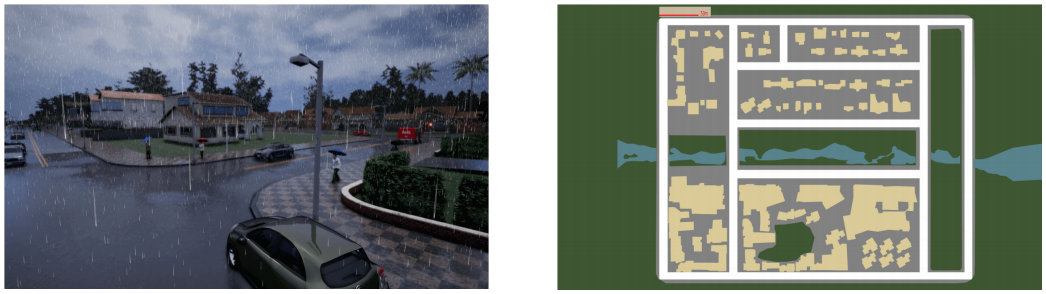


Figura 2.4: Town 1 de Carla

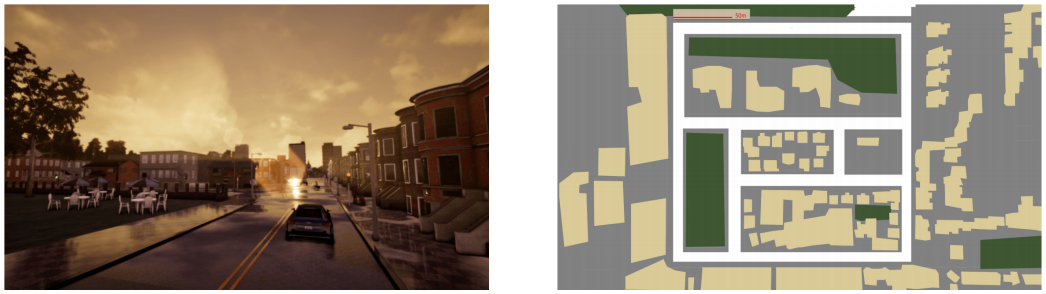


Figura 2.5: Town 2 de Carla

La generación de rutas para los vehículos se realiza a través del PythonAPI, el cual se encarga de suministrar información de carreteras, carriles e intersecciones a los clientes conectados al servidor.

2.1.2.4 Sensores

Los sensores permiten reunir información sobre la simulación de una forma similar a como funcionarían en el mundo real. Carla ofrece una gran cantidad de opciones de configuración (resolución, posición del sensor ...) para que el sistema simulado sea lo mas parecido al real.

Los sensores presentes mas importantes son:

1. Cámaras: el simulador proporciona varios tipos de imágenes procedentes de la cámara delantera, entre las que se encuentran imágenes RGB, mapas de profundidad o segmentación semántica (ver figura 2.6).
2. LiDAR

3. Radar

4. Detector de colisiones

5. Detector de invasión de carril

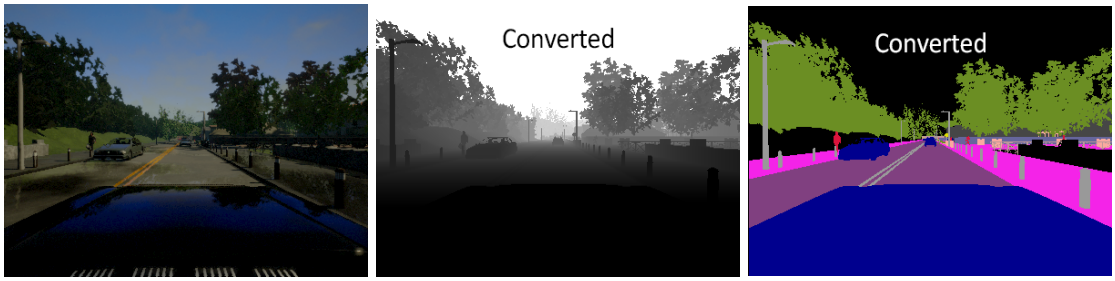


Figura 2.6: Tipos de información de la cámara de Carla (RGB, profundidad y semántica)

2.2 ROS

Robot Operating System (ROS) es un framework compuesto por una colección de herramientas y librerías enfocadas a la creación de sistemas robóticos robustos. En esencia ROS actúa como un sistema operativo ofreciendo servicios a los nodos de ROS, equivalentes a los procesos en un sistema operativo clásico como Windows o Linux.

El intercambio de información entre nodos se realiza a través de los denominados topics, los cuales funcionan como flujos de información con un formato determinado denominado mensaje. En cada topic, un nodo puede publicar información o suscribirse para recibirla. Aparte de los topics, los nodos pueden ofrecer servicios a otros nodos.

La administración de los diferentes nodos la realiza el ROS master, el cual proporciona servicios de identificación y registro al resto de nodos del sistema. Además, lleva el registro de los nodos y de los nodos que publican y se suscriben a cada uno de ellos.

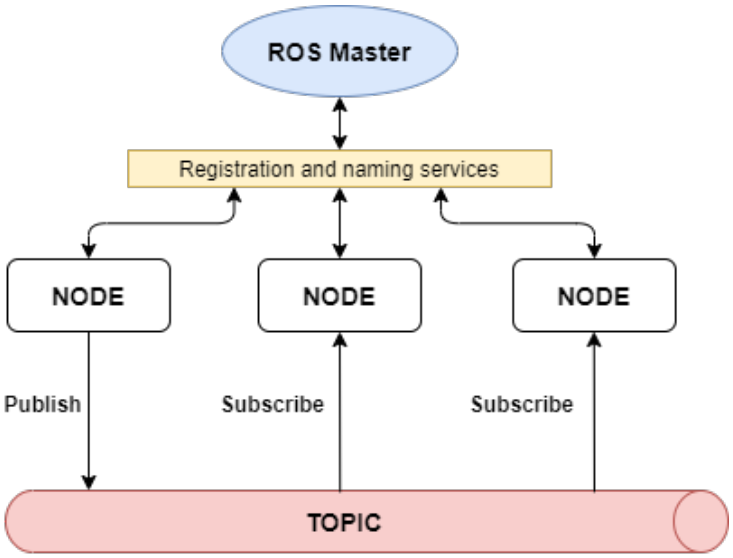


Figura 2.7: Esquema de funcionamiento de ROS

2.3 Docker

Durante el desarrollo de proyectos es habitual ir añadiendo programas, librerías e incluso drivers al software conforme van siendo necesarios. Todas estas tareas de configuración complican y llegan a hacer tediosa la portabilidad del sistema desarrollado. Docker es una herramienta que permite la ejecución de elementos software dentro de entornos aislados, permitiendo omitir todo este proceso.

El trabajo de este proyecto se ha realizado íntegramente sobre Python, utilizando librerías clásicas de Deep Learning como OpenCV 4.2 o Pytorch 1.7. Además, la ejecución se realiza sobre GPU, luego ha sido necesario realizar la instalación de los drivers gráficos de Nvidia y de CUDA. En el caso de querer mover el estimador 3D a otro sistema habría que volver a instalar todo el software requerido o, si hay mas sistemas que utilizan recursos necesarios, habría que realizar pruebas para comprobar que son compatibles.

Para evitar estos problemas, el proyecto T4AC está encapsulado en una imagen de Docker, en la que hay instaladas ciertas versiones del software que son las que se deben utilizar como base para el desarrollo, garantizando así que el conjunto funciona correctamente independientemente del sistema en el que se ejecute.

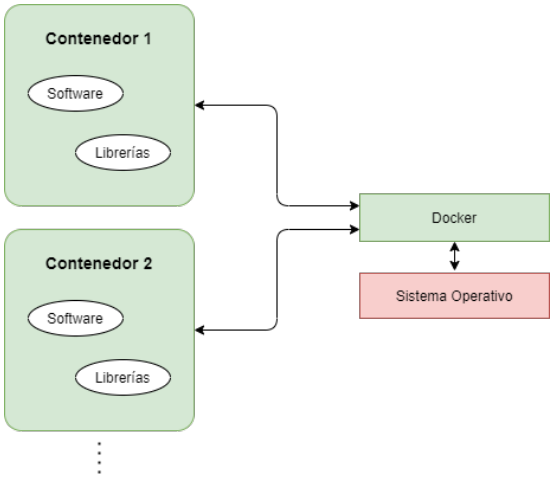


Figura 2.8: Esquema de nivel de abstracción de Docker

Capítulo 3

Estudio de cámara

Como se ha comentado en el capítulo 1, la cámara permite transformar información visual tridimensional de un entorno a 2 dimensiones.

En el caso del sistema implementado en este trabajo, a partir de la información de la cámara se pretende revertir esta transformación estimando la posición real de los objetos utilizando Deep Learning.

3.1 Parámetros de la cámara

A la hora de trabajar con cámaras, existen una serie de valores que reflejan propiedades internas (intrínsecos) y externas (extrínsecos) que son necesarias conocer. Se realizará una explicación de como se va a trabajar con estos datos en el apartado 3.3.

3.1.1 Parámetros intrínsecos

Los parámetros internos o intrínsecos vienen definidos por las características propias de la cámara, entre las que se encuentran:

1. Coordenadas del punto origen proyectadas al plano imagen (u_0,v_0).
2. El factor de distorsión de la cámara (s).
3. Distancia focal, desglosada en ejes x e y (f_x,f_y).

Para facilitar el tratamiento de datos y las transformaciones entre coordenadas, estos parámetros se suelen expresar de forma matricial:

$$K = \begin{pmatrix} f_z & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

La matriz K es esencial para las transformaciones entre coordenadas del mundo y coordenadas en píxel.

3.1.2 Parámetros extrínsecos

Aparte de las propias características de la cámara, en ocasiones es necesario aplicar transformaciones geométricas externas a la cámara para realizar cambios de sistemas de referencias. Estos valores externos se denominan parámetros extrínsecos, y están constituidos fundamentalmente por valores que conforman una rotación (r) y una traslación (t).

Al igual que los parámetros intrínsecos, los parámetros extrínsecos se expresan de forma matricial:

$$E = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}$$

3.2 Sistema de coordenadas

A la hora de trabajar con datos referidos a una cámara debemos tener en cuenta que pueden ser información bidimensional o tridimensional. Es por ello que se establecen dos sistemas de referencia:

1. Para trabajar con las imágenes (información bidimensional) se utiliza un sistema de coordenadas de píxeles. El número de píxeles en cada dimensión define la resolución de la imagen.

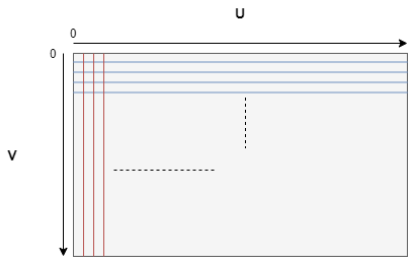


Figura 3.1: Sistema de representación en píxeles de una imagen

El valor que se almacena en cada píxel depende del tipo de imagen con el que se esté trabajando. Los mas comunes son imágenes en escala de grises, con píxeles con valores entre 0 y 1 o 0 y 255, o imágenes RGB, con píxeles con colores definidos en base a tres valores que representan el nivel de rojo, verde y azul. Habitualmente el formato RGB se suele representar como la combinación de tres capas diferentes (R, G, B) como se observa en la figura 3.2.

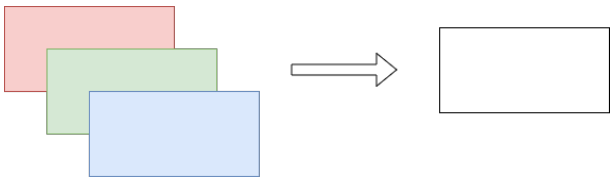


Figura 3.2: Sistema de representación en píxeles de una imagen

2. Para sistemas que requieran referenciar objetos respecto a una cámara en espacios tridimensionales, se define un sistema de referencia de coordenadas cartesianas, con los ejes definidos como se especifica en la figura 3.3.

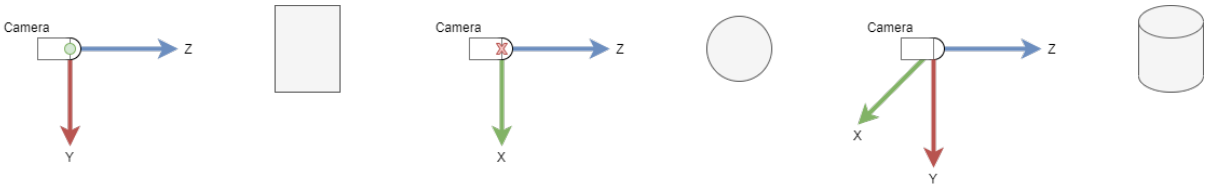


Figura 3.3: Sistema de coordenadas de objetos 3d respecto a la cámara. Vista lateral a la izquierda, vista de pájaro en medio y vista oblicua a la derecha

3.3 Proyecciones y cambios de ejes de referencia

Conociendo los parámetros intrínsecos de la cámara es posible transformar coordenadas tridimensionales en puntos píxel y viceversa, una transformación fundamental para el sistema implementado en este trabajo (capítulo 6) ya que se trabaja en ambos formatos.

Los cálculos para realizar estas transformaciones se realizan de forma matricial, siendo u y v las coordenadas en píxeles y X, Y y Z las coordenadas tridimensionales.

$$\begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = KE \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} f_z & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Si se toma la cámara como el centro del mundo 3D y sin rotación, la matriz de extrínsecos se puede omitir, simplificando el proceso:

$$\begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = \begin{pmatrix} f_z & s & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Un apunte a realizar es que las coordenadas deben estar en formato homogéneo, en el cual un punto bidimensional se define con tres valores como se refleja en las ecuaciones. Para hallar las coordenadas euclídeas (u,v) en píxeles basta con dividir los dos primeros valores por el número que se ha añadido.

$$\begin{pmatrix} wu \\ wv \\ w \end{pmatrix} -> \begin{pmatrix} \frac{wu}{w} \\ \frac{wv}{w} \end{pmatrix} -> \begin{pmatrix} u \\ v \end{pmatrix}$$

Todas estas transformaciones se van a aplicar para obtener proyecciones sobre las imágenes de objetos 3D que hayan sido estimados por el sistema. Además, para almacenar los resultados se ha realizado un cambio de coordenadas tridimensionales para adaptarse a las coordenadas de Carla, en las que el eje vertical pasa a ser el eje Z con sentido opuesto al de cámara y los ejes X y Z pasan a ser Y y X respectivamente (figura 3.4).

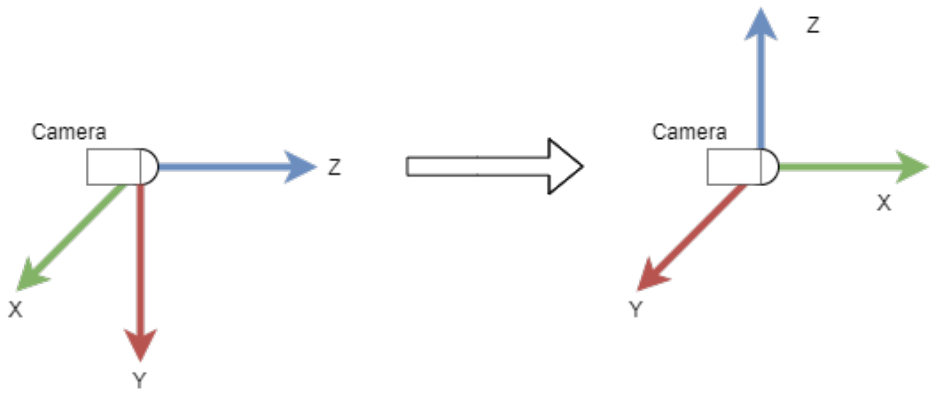


Figura 3.4: Cambio de ejes de referencia de cámara a Carla

Capítulo 4

Estudio del estado del arte de detección 2D y 3D

4.1 Detectores 3D

Gracias al desarrollo tecnológico de la última década, las técnicas de detección basadas en Deep Learning han pasado a ser un pilar fundamental de la visión artificial.

Existen una gran variedad de técnicas de detección en función de los sensores que se utilicen para captar la información como PointPillars [3], para extraer información de una nube de puntos de LiDAR, o redes de detección en 2D sobre imágenes, como RCNN y sus variantes FAST [4] y FASTER [5] o la familia YOLO.

La mayoría de las publicaciones y artículos de detección tridimensional utilizan un enfoque basado en fusión de la información de la cámara con los datos de otros sensores como LiDAR o radar, que aportan información en 3D más fiable, obteniendo así detecciones más robustas y aprovechando los puntos fuertes de cada sensor. En nuestro caso, se pretende trabajar únicamente con información proveniente de la cámara, luego se han estudiado varios ejemplos de publicaciones que sigan esta línea.

4.1.1 Expandable YOLO

You Only Look Once (YOLO) es uno de los modelos de detección 2D más influyentes de los últimos años. En el paper de Takahashi [6] se plantea utilizar una YOLOv3 como base de un sistema de extracción de características tridimensionales a la que denomina Expandable YOLO (E-YOLO).

Cualquiera de las redes de la familia YOLO admite como entrada una imagen RGB (con dimensión H x W x 3) de la que se extrae la información a partir de los colores. En el caso de E-YOLO a esas capas se le añade la información de profundidad obteniendo una entrada de 416 x 416 x 4.

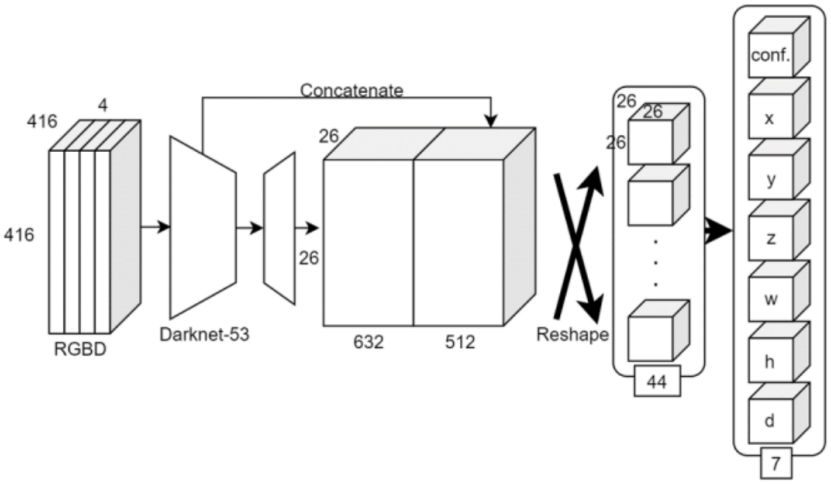


Figura 4.1: Estructura de la red E-YOLO

En la figura 4.1 se propone utilizar una Darknet-53 para la extracción de características, al igual que se realiza en YOLOv3, solo que en este caso se deben extraer del color y profundidad simultáneamente. A la salida de la Darknet se le concatena una salida de una capa intermedia y se le aplica un reshape para pasar a una información tridimensional. A esta información se le aplican varias capas de convolución 3D para llegar a la salida de 10 canales: ocho canales para la información 3D que proporcionan dos scores de la detección, la posición del objeto (x,y,z) y sus dimensiones (h,w,l). Para la selección de la detección óptima entre los candidatos a la hora de realizar la inferencia, se utiliza el método de Non Maximum Suppression, explicado en A.1, en el que se realiza un filtrado de las bounding box en función de su score y su IoU3D para garantizar que únicamente hay una detección por objeto.

Respecto al entrenamiento de la red, se ha utilizado un dataset proporcionado por la universidad de Chuo a partir de imágenes en un aula con alumnos [7], con el que se obtienen los siguientes resultados únicamente para la clase Pedestrian:

	2D IoU	3D IoU
Mean (Pedestrian)	0.54	0.39
Max (Pedestrian)	0.92	0.85

Tabla 4.1: Resultados E-YOLO

De media se obtiene un IoU3D de alrededor de un 40 %, el cual para ser un sistema con información únicamente proveniente de una cámara es aceptable. Cabe destacar que al añadir la información de profundidad y realizar el entrenamiento ha provocado que el IoU2D medio haya disminuido respecto a la YOLO normal.

Por último hay que destacar que el sistema responde con una velocidad de 44 FPS funcionando sobre una Nvidia GTX 1080Ti, una tasa bastante buena que permitiría acoplarse sin problemas en un sistema mas complejo. Para comparar, una YOLOv3 sobre el mismo hardware responde a 74 fps, luego es una pérdida de rendimiento considerable.

4.1.2 Realtime 3D Object Detection for Automated Driving Using Stereo Vision and Semantic Information

El método explicado en el apartado 4.1.1 utiliza un enfoque end-to-end basado en deep learning completamente para realizar la detección tridimensional. En el caso del trabajo de Hendrik Königshof [8] se propone un sistema compuesto por varias etapas, combinando redes convolucionales con varios procesamientos de imágenes y datos para realizar la misma tarea.

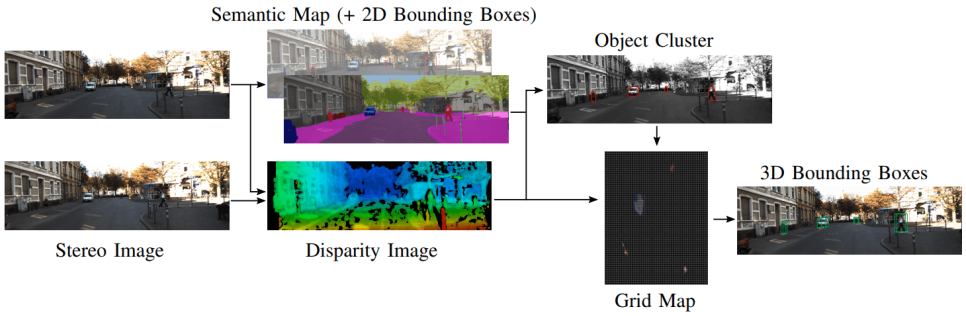


Figura 4.2: Estructura del sistema 3D con información semántica

Como se puede observar en la figura 4.2, partiendo de la imagen capturada por la lente izquierda se obtiene una segmentación semántica y una detección opcional con boundig box. Al mismo tiempo un algoritmo de block matching [9] genera una estimación de la disparidad entre las imágenes de la cámara estéreo, lo que permite obtener información sobre la profundidad de los elementos de la imagen. Con estos datos se realiza un clústering sobre las disparidades basado en Etiquetado de Componentes Conectados, en el que se tiene en cuenta si los píxeles son de la misma clase (proporcionada por la segmentación). A partir de los clústers obtenidos se proyectan los puntos a coordenadas mundo y se extraen los bordes y se genera la bounding box 3d teniendo en cuenta la clase del objeto.

Para la detección de objetos sobre la imagen de la lente izquierda se utiliza una Convolutional Neural Network (CNN) con la siguiente estructura basada en [10]:

1. Un backbone basado en ResNet-38 [11] cuya salida alimenta dos cabezas.
2. Una cabeza que utiliza la salida de ResNet para obtener una segmentación semántica con la misma resolución que la imagen original.
3. Una cabeza que realiza detección y regresión de bounding box en 2D.

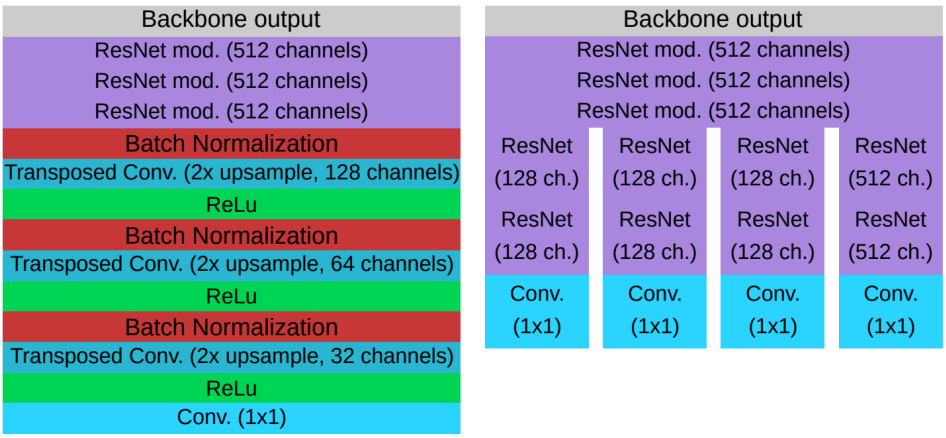


Figura 4.3: Estructura de las cabezas para detección

El entrenamiento de la CNN se ha realizado sobre el dataset Cityscapes [12] y parte del dataset Kitti [13], del que también se extraen los datos de validación.

Para la evaluación se han utilizado las métricas presentes en el evaluador de Kitti, evaluando el Average Precision para las tareas de detección 2D, 3D y vista de pájaro utilizando los valores límite de IoU del benchmark de Kitti (70 % para coches y 50 % para el resto).

Class	Benchmark	Easy	Moderate	Hard
Car	2D Detection	0.5756	0.4892	0.4281
	3D Detection	0.2850	0.2410	0.2032
	Bird's Eye View	0.5932	0.4948	0.4316
Pedestrian	2D Detection	0.4454	0.3201	0.3150
	3D Detection	0.0427	0.0425	0.0426
	Bird's Eye View	0.0539	0.0530	0.0519
Cylclist	2D Detection	0.2373	0.1766	0.1194
	3D Detection	0.0662	0.0663	0.0403
	Bird's Eye View	0.0770	0.0759	0.0751

Tabla 4.2: Resultados sistema con información semántica

Al contrario que la red explicada en el apartado 4.1.1 la cual está orientada detección en personas en interiores, la red planteada en este apartado está pensada para trabajar sobre mas clases en conducción autónoma. Los resultados del evaluador muestran una detección 3D en coches mejorable (entre 20 % y 28 %), mejorando en vista de pájaro, sin embargo en 2D son bastante pobres comparándolos con otros detectores 2D del estado del arte que pueden llegar a alcanzar valores de AP de 0.95. El resto de clases presentan resultados bastante por debajo de los obtenidos para los coches.

Por último, hay que destacar que el tiempo medio de procesado del sistema sobre Kitti en una NVIDIA TITAN X es de 79 ms (13 FPS) si se incluye la información de las bounding-box permitiendo paralelizar y acelerar el proceso de clústering, o de 92 ms (11 FPS) si no se incluye. Estos tiempos pueden variar en función de la cantidad de objetos a procesar, luego son datos orientativos.

4.1.3 3D Bounding Box Estimation Using Deep Learning and Geometry

Al contrario que la red del apartado 4.1.1, que se basa por completo en deep learning; o del sistema 4.1.2, que utiliza deep learning junto con información de la disparidad y post-procesado; el detector propuesto por Arsalan Mousavian en [14] plantea utilizar varias etapas con deep learning a las que se le añade información de geometría para llegar a la detección final.

En rasgos generales, en la publicación [14] se realiza la estimación tridimensional a partir de una única imagen (luego no es necesaria una cámara estéreo) junto a las detecciones en 2D con bounding-box de los objetos a estimar. Es por esto que el sistema cuenta con dos etapas:

1. Detector 2D: la primera fase del sistema es la detección de objetos con bounding-box en 2D lo más precisas posibles. Para esta etapa se puede usar cualquier detector 2D del estado del arte, aunque habría que priorizar los que presentan mejores resultados para evitar introducir errores en la segunda etapa.

2. Estimador 3D: realiza una regresión de las dimensiones de los objetos y de la orientación local con la estructura reflejada en la figura 4.4. Aplicando un post-procesado basado en geometría se puede calcular las coordenadas reales de la bounding-box 3D y orientación global.

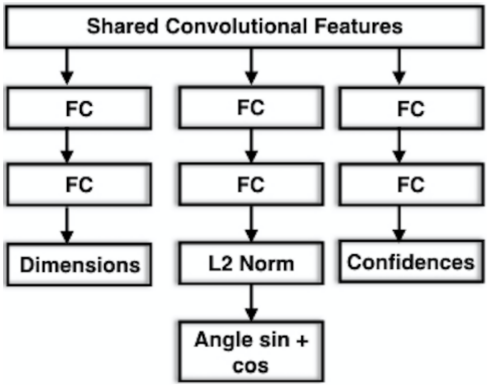


Figura 4.4: Estructura de la red volumétrica

Este sistema será la base elegida para implementar nuestro trabajo, luego se realizará una explicación mas en profundidad en el capítulo 6.

Los resultados de la evaluación del AP sobre Kitti son los mostrados en la tabla 4.3. En la publicación no especifican resultados de la clase Pedestrian.

Class	Easy	Moderate	Hard
Car	0.9298	0.8904	0.7717
Cyclist	0.8394	0.7416	0.6484

Tabla 4.3: Resultados red volumétrica

Respecto al tiempo de ejecución, el autor asegura que de media se tardan unos 0.4 segundos. Al igual que pasaba en el sistema del apartado 4.1.2, el tiempo es función del número de objetos a procesar además de depender del detector 2D que se utilice.

4.2 Elección del detector

Una vez planteadas varias alternativas, se ha elegido una de ellas para utilizar como base en el trabajo desarrollado conforme al siguiente razonamiento:

1. Estructura del sistema: cada uno de los sistemas propuestos plantea una línea de ejecución diferente.

   El sistema E-YOLO (4.1.1) plantea una red neuronal end-to-end, la cual puede verse como un bloque único en el que realizar ajustes o modificaciones es menos intuitivo y puede influir muy notablemente en los resultados. Cabe destacar que el entrenamiento se ha realizado con imágenes de interiores, luego para aplicar este modelo en conducción autónoma sería obligatorio un re-entrenamiento con nuevos datos.

   Por otra parte, el sistema que utiliza segmentación semántica (4.1.2) plantea un modelo en tres etapas: en la primera se utiliza deep learning y block matching de forma paralela para realizar las

detecciones y la disparidad. Posteriormente se realiza la clusterización para calcular, en el último tramo, la bounding box 3D.

Por último la red volumétrica (4.1.3) plantea un sistema en dos etapas: una primera detección con bounding box en 2D y una red volumétrica para obtener las dimensiones y orientación de las bounding box 3D para su posterior proyección.

2. Métricas y resultados: la E-YOLO (4.1.1) ha sido entrenada y evaluada sobre un dataset de imágenes en un aula propio de los creadores, entorno que no es extrapolable a conducción autónoma. Aún siendo así, los resultados en 2D son bastante mejorables con un mIoU de 0.54, mientras que 3D se obtiene un mIoU3D de 0.39. Los otros dos sistemas se han evaluado sobre el dataset Kitti, y ambos presentan buenos resultados, con valores de AP para la clase 'Car' entre 50 y 90 %.

3. Tiempos de ejecución: si bien los datos ofrecidos por las publicaciones se han realizado sobre Graphics Processing Unit (GPU) diferentes y pueden variar en función de la escena, es bastante evidente que la red volumétrica (4.1.3) es la que tiene un tiempo de procesado mayor.

De estos datos se puede concluir que la red volumétrica sacrifica tiempo de procesado para conseguir unos resultados superiores a los otros dos sistemas, presentando una arquitectura en dos fases en lugar de ser un único bloque. Se decide que se utilizará como base para el proyecto dado que ofrece un framerate suficiente y buenos resultados.

Cabe destacar que en el sistema de percepción del grupo T4AC (1.3) se requieren detecciones en 2D para otras tareas de detección. Es por esto que se va a realizar un breve estudio de estado del arte en detección 2D para elegir qué detector utilizar en el proyecto, evitando así tener varios sistemas realizando la misma tarea, con el coste computacional que supondría.

4.3 Detectores 2D

La detección 2D es una de las tareas más extendidas en el campo de la visión artificial. En este apartado se pretende proporcionar una idea general de los detectores más utilizados en los últimos años, entre los que se encuentran las redes YOLO o EfficientDet, como se muestra en las figuras 4.5 y 4.6.

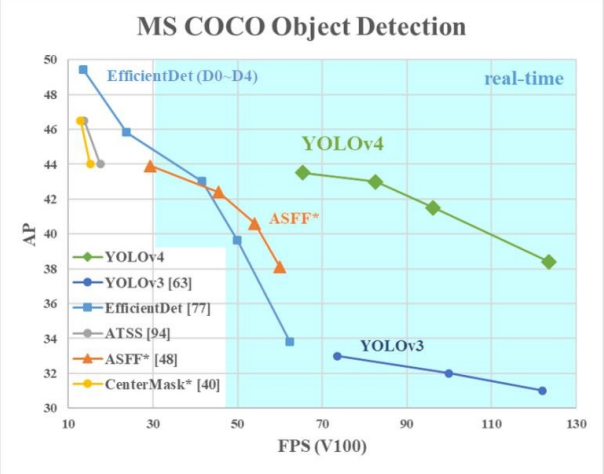


Figura 4.5: Comparación de resultados y tiempos de ejecución sobre COCO val2017 con diferentes redes sobre una NVIDIA V100

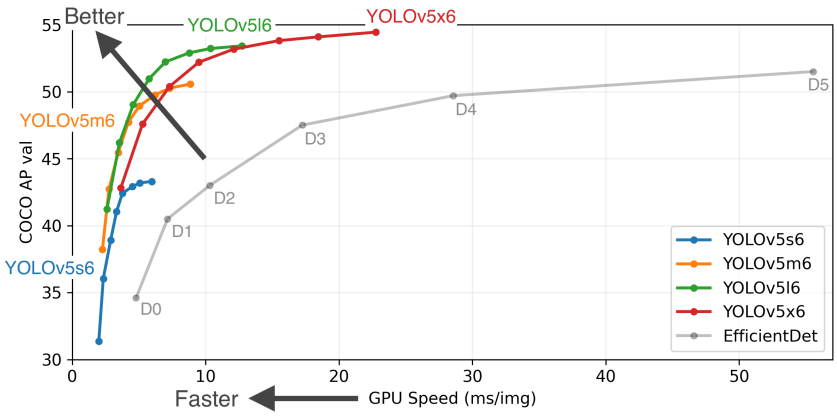


Figura 4.6: Comparación de resultados y tiempos de ejecución sobre COCO val2017 de las redes YOLOv5 sobre una NVIDIA V100

4.3.1 YOLO

Las redes de la familia YOLO son los modelos mas utilizados para tareas de detección debido a sus buenos resultados y tiempos de procesamiento. En el año 2016 salió a la luz la primera versión de YOLO [15], revolucionando el estado del arte de la detección 2D. Desde ese momento se han ido publicando nuevas versiones, como YOLOv3 [16], YOLOv4 [17] o YOLOv5 [18], que han ido mejorando y puliendo diversos aspectos de sus predecesoras.

Estas redes son detectores de una única etapa o one stage detectors, es decir, realizan detección y clasificación conjuntamente. Además, al contrario que otros detectores, YOLO realiza las detecciones teniendo en cuenta el contexto general de la imagen, no realiza ningún tipo de división en regiones como hace RCNN.

Como se puede observar en las figuras 4.5 y 4.6, la evolución a través de las diferentes versiones ha sido bastante notoria manteniendo siempre un tiempo de ejecución bastante bajo. Se observa como la versión 3 presenta resultados de alrededor de 32 % de AP sobre COCO, resultados mas rápidos pero bastante inferiores a otras redes como EfficientDet. La versión 4 consigue resultados que llegan hasta el 43 %, consiguiendo acercarse a EfficientDet con una mayor velocidad, aunque aún bastante alejada de los mejores valores que ofrece esta última. Por último, la versión 5 da un gran salto en resultados, superando a EfficientDet tanto en velocidad como en AP en las versiones 5m, 5l y 5x.

4.3.2 EfficientDet

Al igual que YOLO, EfficientDet [19] es un one stage detector, realizando la detección en una única fase. La idea sobre la que se desarrolló el detector es la de ofrecer una estructura que cuente con cierto grado de escalabilidad a la vez de mantener una buena eficiencia.

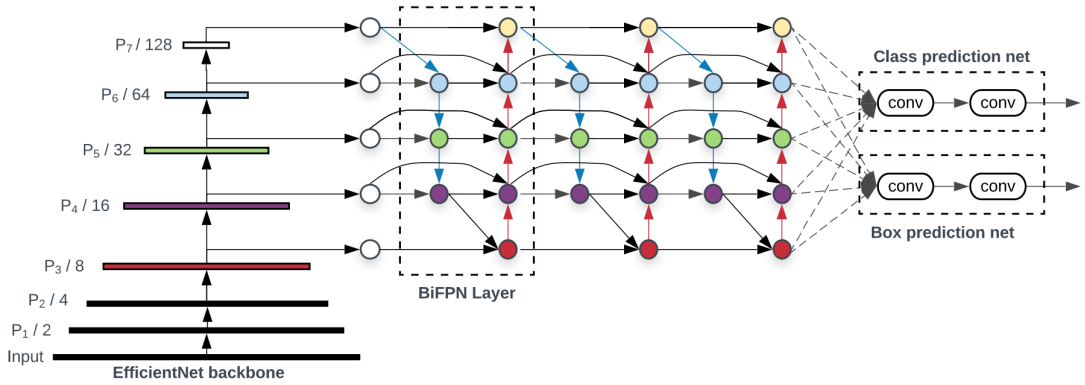


Figura 4.7: Estructura de EfficientDet

Como se puede observar en la figura 4.7 el backbone del detector es una red EfficientNet [20], seguida de una red BiFPN para extraer características y dos redes de predicción de clase y bounding-box. En función del tamaño de la red que se desee utilizar cada componente escala siguiendo los siguientes valores:

	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5
D7x	1536	B7	384	8	5

Figura 4.8: Valores de escalabilidad de EfficientDet

4.3.3 RCNN

Al contrario que las redes YOLO o EfficientDet, RCNN y sus variantes son detectores de dos etapas o two stage detectors, en la que primero se realiza una selección de áreas de interés y posteriormente se realiza una clasificación. Este tipo de detectores suele tener un tiempo de procesado mas elevado que los one stage, pero presentan resultados superiores.

El primer detector de esta familia fue RCNN [21] de Ross Girshick, que planteaba la siguiente línea de detección:

1. Extracción de propuestas de región (unas 2000) por un algoritmo de búsqueda selectiva A.2.

2. Extracción de características de las regiones anteriores utilizando una CNN.

3. Las características extraídas se utilizan para alimentar una Support Vector Machine (SVM), que se encarga de determinar si en cada región existe un objeto de interés.

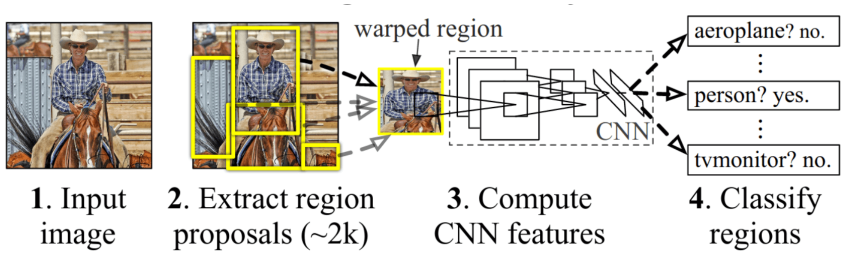


Figura 4.9: Estructura de RCNN

El tiempo de ejecución por imagen de este detector es muy elevado, impidiendo incluso ser implementado en tiempo real. Además, la extracción de características se hace con un algoritmo, luego a la hora de ser entrenado esta etapa no sufre ningún cambio.

Años mas tarde, el creador de RCNN propuso una serie de mejoras que ayudaron a mejorar el funcionamiento considerablemente, dando lugar a Fast-RCNN [4].

En este detector en lugar de alimentar a la CNN con las regiones, se le alimenta con la imagen completa para extraer las características. A partir de estas se generan las propuestas de regiones utilizando de nuevo un algoritmo de búsqueda selectiva, que se alimentan a su vez a una capa softmax para determinar si hay un objeto de interés.

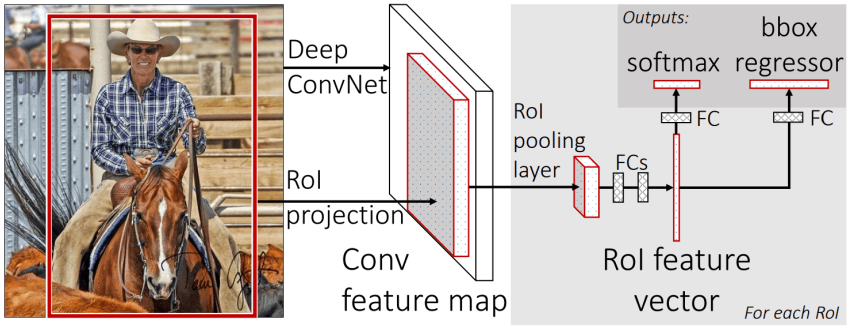


Figura 4.10: Estructura de Fast-RCNN

	Fast R-CNN			R-CNN		
	S	**M**	**L**	**S**	**M**	**L**
train time (h)	**1.2**	2.0	9.5	22	28	84
train speedup	**18.3**×	14.0×	8.8×	1×	1×	1×
test rate (s/im)	0.10	0.15	0.32	9.8	12.1	47.0
▷ with SVD	**0.06**	0.08	0.22	-	-	-
test speedup	98×	80×	146×	1×	1×	1×
▷ with SVD	169×	150×	**213**×	-	-	-

Figura 4.11: Comparación de tiempos entre RCNN y Fast-RCNN

Si bien Fast-RCNN supuso una mejora considerable de rendimiento respecto a RCNN, utilizar un algoritmo para la generación de propuestas de región introduce un gran retardo en el proceso, además de no beneficiarse de los posibles entrenamientos.

Con esto en mente Shaoqing Ren ideó Faster-RCNN [5]. Este detector está compuesto por Fast-RCNN al que se le añade un bloque anterior para la generación de propuestas de región, llamado Region Proposal

Network (RPN). Este bloque RPN toma como entrada la imagen completa para alimentar una CNN para la extracción de características. Con esta información, en lugar de utilizar un algoritmo para la generación de las propuestas, se utiliza una red adicional para predecir las propuestas de región. Eliminando el uso de un algoritmo mejora considerablemente la velocidad del detector.

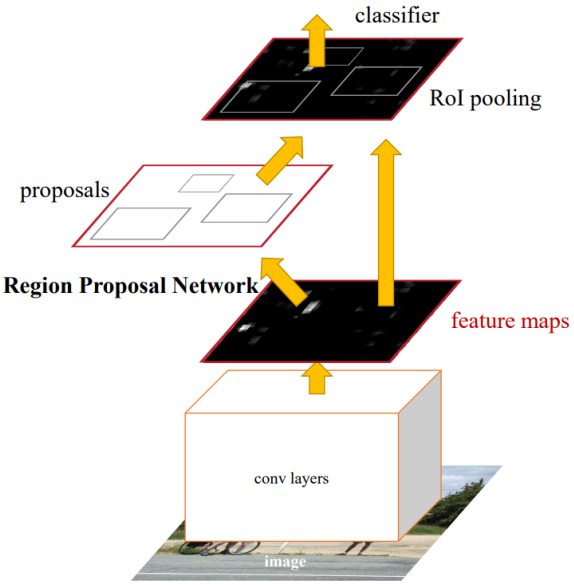


Figura 4.12: Estructura de la red RPN de Faster-RCNN

Capítulo 5

Kitti dataset

El dataset Kitti [13] lleva siendo un referente en tareas de visión artificial en conducción autónoma desde que se presentó en el CVPR del año 2012. La información está generada en base a los siguientes sensores:

1. Una cámara estéreo que proporciona imágenes en color y escala de grises. Modelo Point Grey Flea 2 (FL2-14S3M-C).
2. Un lidar que proporciona una nube de puntos de distancia a elementos del entorno con una gran precisión. Modelo Velodyne HDL-64E.
3. Un módulo GPS inercial para la localización. Modelo OXTS RT 3003.

Además de la información en crudo, se proporcionan una serie de benchmarks para evaluar sistemas en diferentes tareas con sus respectivas métricas.5.2

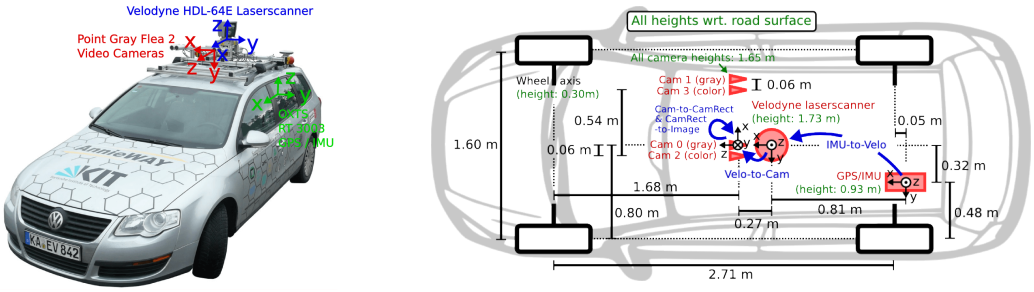


Figura 5.1: Esquema del setup de sensores de Kitti

5.1 Estructura

Kitti proporciona datos en función de la tarea sobre la que se desee trabajar, entre las que se encuentran odometría visual, tracking o detección con diferentes sensores y métodos.

En el trabajo desarrollado se va a emplear información de la cámara para realizar la estimación de la posición de los objetos. Se ha utilizado el apartado de object detection, del que se han extraído las imágenes de la lente izquierda de la cámara estéreo junto al correspondiente etiquetado y parámetros de la cámara.

Cabe destacar que se utiliza la lente izquierda ya que es la que normalmente se toma como referencia en sistemas con cámara estéreo.

El etiquetado de cada imagen está guardado en un archivo de texto con el mismo nombre, en el que cada línea se corresponde a un objeto etiquetado diferente y del que se proporciona:

1. Type: un string que especifica a que tipo de objeto se refiere. Los tipos más comunes son ´Car´, ´Pedestrian´ y ´Cyclist´.
2. Truncated: un número float de 0 a 1 que especifica si el objeto en cuestión se sale del límite de la imagen.
3. Occluded: indica si el objeto está oculto.
4. Alpha: ángulo de observación del objeto, que se corresponde con el ángulo que forma la línea que une la cámara con el centroide del objeto con el eje X en coordenadas cámara.
5. Bounding Box: cuatro valores de píxeles que definen left, top, right, bottom de la bounding box en 2D de cada objeto.
6. Dimensions: tres valores que especifican las dimensiones tridimensionales del objeto (altura, anchura y longitud).
7. Location: tres valores que definen la posición xyz del centroide del objeto.
8. Rotation y: valor del ángulo que forma la orientación del objeto con el eje X en coordenadas de la cámara.

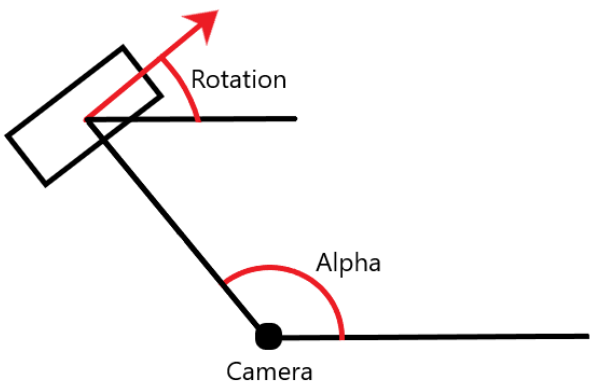


Figura 5.2: Ángulos alpha y orientación de Kitti

Esta estructura de etiquetado también se debe seguir a la hora de almacenar los resultados de nuestro sistema, añadiendo un campo mas en el que se almacena el score de cada detección. Esto es importante para posteriormente aplicar el evaluador propio del dataset. De esta manera Kitti proporciona información de los elementos de sus imágenes tanto en dos como en tres dimensiones, pudiendo utilizar además la matriz de parámetros de cámara para transformar entre coordenadas reales y puntos de la imagen si fuera necesario.

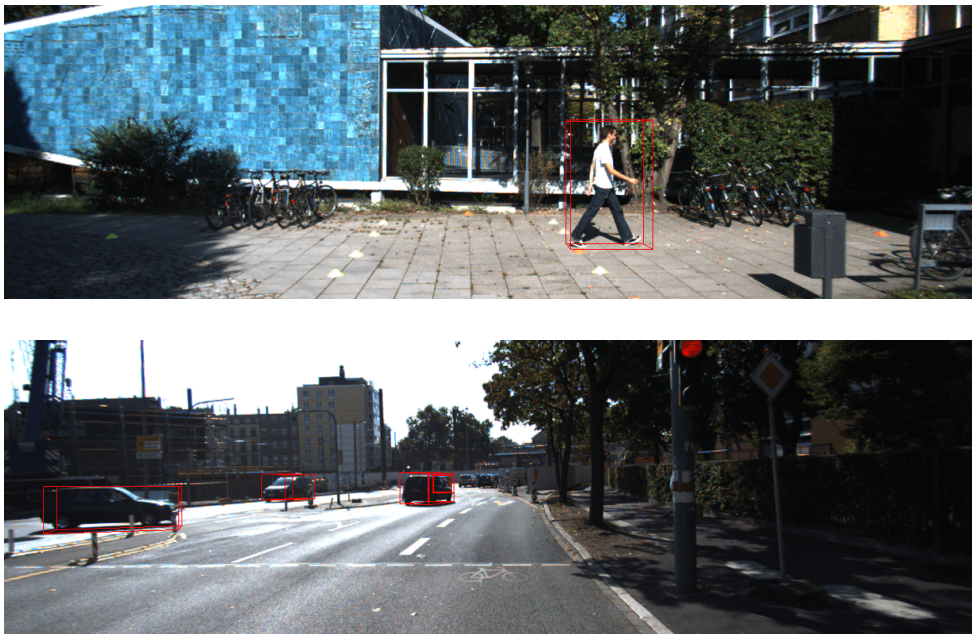


Figura 5.3: Imágenes de Kitti con el ground truth representado.

5.2 Evaluador

Además de los datos de los sensores etiquetados, Kitti proporciona un kit de desarrollo para trabajar mas cómodamente con los datos y para evaluar resultados. Este evaluador analiza los resultados del sistema implementado sobre los datos del dataset y genera unas métricas para cada clase a detectar en función de la dificultad. Los valores generados para cada clase se basan en Average Precision, métrica utilizada para reflejar el rendimiento respecto a positivos y negativos en evaluación de detecciones. A partir del rendimiento en la detección se calcula el Average Orientation Similarity multiplicando el AP por el coseno de la diferencia media entre las detecciones y el ground truth.

Si bien en el kit de desarrollo se proporcionan las herramientas necesarias para la evaluación, estas se encuentran en C++, mientras que nuestro proyecto se ha realizado en Python. Es por esto que se ha utilizado un repositorio de GitHub [22] que a partir del dev-kit original ya incorpora el código necesario para evaluar directamente los archivos de texto sin necesidad de integrar todo el código en el proyecto.

Cabe destacar que las clases evaluadas son 'Car', 'Pedestrian' y 'Cyclist', en las que un true positive (una detección es asociada con un objeto del ground truth) es considerado si el IoU3D es mayor al 70 % para el caso de coches y al 50 % para peatones y ciclistas.

Capítulo 6

Estimador 3D

Como se ha especificado en el cap 4, se va a utilizar la red volumétrica [14] para extraer las propiedades tridimensionales de los objetos capturados por la cámara. Este sistema necesita una etapa previa que le proporcione detecciones bidimensionales, luego se ha dividido en dos etapas diferentes.

6.1 Detección 2D

Para poder extraer las características tridimensionales primero es necesario realizar una detección de los elementos de interés sobre la imagen de la cámara.

De los detectores estudiados en el apartado 4.3 se ha decidido utilizar el detector Yolov5 [18] que ofrece uno de los mejores rendimientos actuales en detección en imágenes. Como refleja la gráfica 4.6, existen varios modelos dentro de Yolov5 con diferentes tamaños y eficiencias, todos entrenados sobre el dataset COCO. Para lograr un compromiso entre resultados y tiempo de procesado se ha decidido utilizar la versión Yolov5L.

6.2 Estimación 3D

Una vez realizadas las detecciones 2D entra en juego la red volumétrica [14] para extraer las características 3D.

La idea de la que parte este sistema es que la bounding box 3D se ajusta perfectamente a la forma de la 2D, y por lo tanto, cada borde de la bidimensional debe coincidir con una proyección de la tridimensional.

Para realizar la estimación de la bounding box 3D se debe aplicar regresión a varios parámetros clave para la caracterización de un cubo tridimensional. El primero de ellos sobre el que se trabaja es la orientación sobre cada eje, especialmente importante para el calculo de los diferentes vértices y proyecciones. Además de la orientación es necesario contar con mas variables para poder determinar por completo las propiedades de las bounding box. Se podría trabajar con la traslación (posición) del objeto, pero se utilizan las dimensiones ya que son valores mas uniformes y por lo tanto mas sencillos de determinar mediante regresión.

Como se representa en la figura 6.1 se pueden definir varios ángulos para caracterizar a una bounding box tridimensional si únicamente se trabaja con rotaciones respecto al eje Y (vertical):

1. Ángulo de observación (θray): ángulo que forma la línea que une el centro del objeto con la cámara con el eje X.
2. Orientación global (θ): ángulo que forma el vector perpendicular a la parte delantera del objeto (dirección del objeto) con el eje X.
3. Orientación local (θ_l): ángulo que forma el vector dirección del objeto con la línea que une la cámara al objeto.

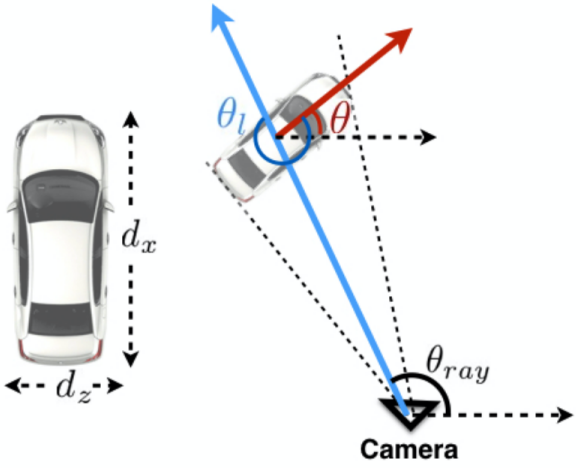


Figura 6.1: Representación de ángulos de la red volumétrica (ángulo de observación en negro, orientación global en rojo y orientación local en azul)

La orientación global es la requerida para la construcción de la bounding box 3D. En lugar de aplicar regresión directamente a este ángulo, el sistema estima la orientación local y a partir de la información de la imagen se calcula de una manera sencilla el ángulo de observación. Combinando estos valores se obtiene el valor de orientación global que se busca. Realizando este calculo en lugar de una regresión directa se evitan los problemas asociados a la propia perspectiva de la cámara. Por ejemplo, si un vehículo externo nos adelanta por la izquierda pudiera parecer que la orientación global cambia, sin embargo, únicamente cambian la orientación local y angulo de visión, que combinados proporcionan una orientación global constante.

Una vez obtenidas la orientación y dimensiones del objeto sobre el que se está trabajando, se debe obtener la posición para completar toda la información necesaria. Cada lado de la bounding box 2D permite construir una ecuación para los valores $x_{min}, x_{max}, y_{min}, y_{max}$ con la siguiente forma matricial:

$$value = \left(K \begin{bmatrix} I & RX_o^i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} T_x \\ T_y \\ T_z \\ 1 \end{bmatrix} \right)$$

Siendo I una matriz identidad de 3x3 y X_o^i las coorenadas 3D del vértice número i. Con las 4 ecuaciones se puede contruir un sistema de ecuaciones para despejar los valores de traslación.

El proceso se complica en el momento en el que el sistema no sabe qué vértice 3D tiene su proyección en cada lado de la bounding box 2D. Por este motivo es necesario resolver este sistema de ecuaciones varias veces y elegir los valores tridimensionales que minimizan el error al ser proyectados de nuevo sobre la imagen. Este proceso de resolución se realiza de forma paralela, y en el peor de los casos debe realizarse

4096 veces. En el caso de considerar únicamente las posiciones realistas de los objetos (por ejemplo un coche no puede sostenerse en el suelo sobre un lateral, siempre es por las ruedas), este valor desciende hasta 256.

Con este proceso completado ya se han obtenido los parámetros necesarios para caracterizar la bounding box 3D por completo. Esta información es la que se enviará al resto del proyecto T4AC a través de ROS, como se comentará en el apartado 7. En el caso de querer representar los resultados sobre la imagen original se realiza lo siguiente:

1. Se calculan las coordenadas tridimensionales de los ocho vértices a partir de la anchura, altura y longitud. Estas coordenadas se construyen sobre el origen, es decir, como si el objeto tuviera su centro en $(0, 0, 0)$.
2. Se aplica una rotación para que el objeto tenga su orientación global correspondiente.
3. Se aplica una traslación para llevar la caja desde el origen al centro real del objeto.
4. Se calculan las proyecciones al plano imagen de los ocho vértices utilizando la matriz intrínseca.
5. Se dibuja sobre la imagen original los vértices proyectados y las líneas de unión entre ellos.

Uno de los grandes problemas de este sistema es la gran variedad que puede sufrir el tiempo total de ejecución. El tiempo que se tarda en generar la bounding box es la suma de dos redes diferentes, la Yolov5 se mantiene prácticamente constante y con un tiempo bastante inferior a la volumétrica, mientras que esta última depende en gran medida del número de objetos a estimar, llegando a ofrecer 10 FPS en los peores casos.

Por otra parte, los resultados están fuertemente ligados a la propia posición real del objeto. Como se observa en la figura 6.2, a partir de cierta distancia la red estimadora comienza a tener unos valores de error que hacen que los resultados no sean válidos. Además, si los objetos se encuentran en los bordes del campo de visión de la cámara pueden sufrir rotaciones y traslaciones indeseadas. Para minimizar este efecto se realizan varios filtrados a lo largo del proceso de detección:

1. Las detecciones 2D procedentes de la Yolov5 que tengan alguno de sus bordes en los límites laterales de la imagen se eliminan. Realizando esto se evita que la volumétrica procese objetos cortados.
2. Una vez obtenidas las posiciones finales de los objetos se realiza un filtrado si la distancia a la cámara es superior a un umbral.

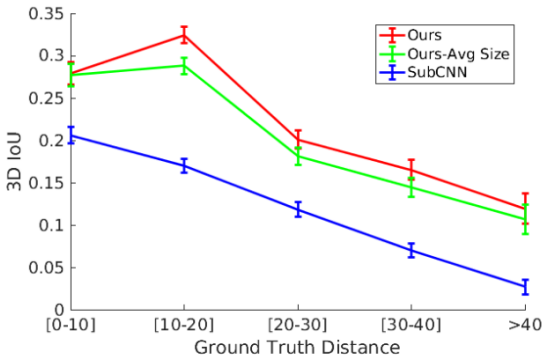
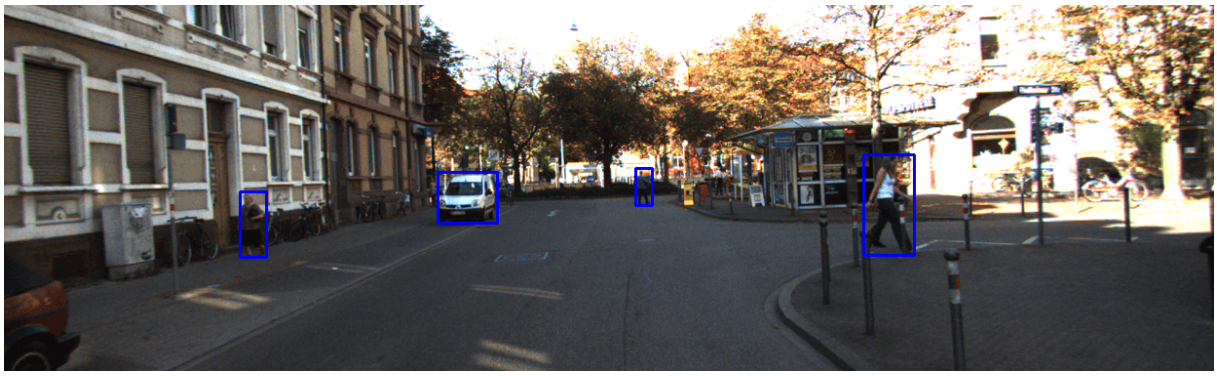


Figura 6.2: Evolución del 3D IoU con la distancia sobre la clase Car de Kitti

En las imágenes de la figura 6.3 se representan las etapas por las que va pasando la imagen que se está procesando. En primer lugar se muestra la imagen de la cámara (a), posteriormente se consigue la detección 2D de la Yolo para pasar por la red volumétrica (b) y estimar sus características 3D (c).

(a) Imagen de la cámara



(b) Detección 2D

(c) Estimación 3D

Figura 6.3: Fases de detección en el sistema estimador 3D

Capítulo 7

Implementación en el proyecto Techs4AgeCar

Como se ha explicado en el capítulo 1, el sistema del proyecto T4AC se encuentra implementado en una imagen de Docker y funcionando sobre paquetes de ROS. En el caso de la red volumétrica de este trabajo se ha incluido en la capa de percepción, creando un nodo de ROS y varios archivos con funciones necesarias para el proceso de detección.

En la figura 7.1 se ha plasmado la estructura de los nodos y topics que afectan al estimador tridimensional.

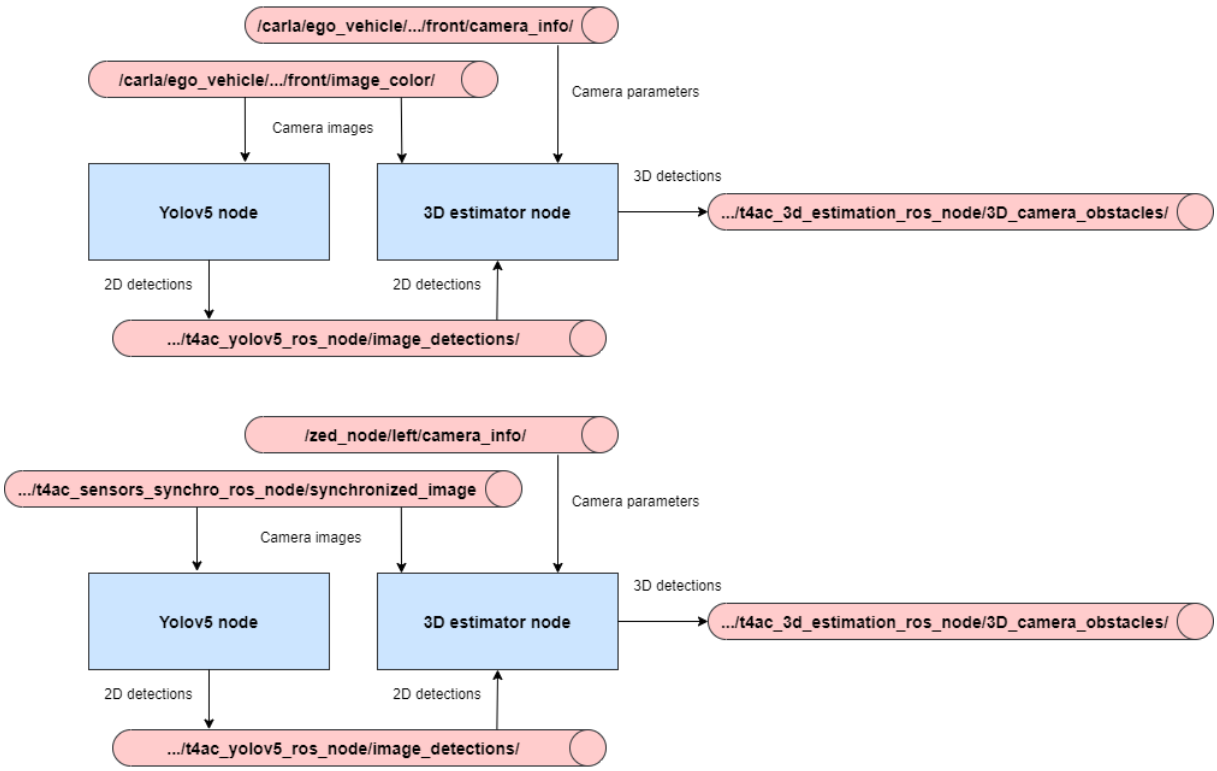


Figura 7.1: Esquema del funcionamiento del estimador en ROS en simluación (arriba) y en real (abajo)

7.1 Creación de nodos de ROS y uso de topics

Para la realización de la estimación tridimensional ha sido necesario integrar dos nodos de ROS diferentes, uno por cada etapa del estimador 3D.

El primero de ellos se encarga de ejecutar la Yolov5L para proporcionar las detecciones en 2D a la red volumétrica. Al comenzar la ejecución el nodo se encarga de suscribirse al topic del que se van a sacar las imágenes (*/carla/ego_vehicle/.../front/image_color/* si es sobre simulación o *.../t4ac_sensors_synchro_ros_node/synchronized_image/* si se trabaja sobre real) y de cargar la Yolov5 en GPU. Posteriormente el nodo va adquiriendo imágenes del topic y ejecutando la Yolo que se ha cargado anteriormente obteniendo las respectivas detecciones bidimensionales. Estas detecciones se publican en el topic */t4ac/perception/.../t4ac_yolov5_ros_node/image_detections/* en un mensaje de ROS con la siguiente estructura, a la que se le añade una cabecera con información extra como el timestamp:

1. Type: un string que identifica que tipo de objeto es.
2. Score: un valor de 0 a 1 que identifica la confidencia de la detección.
3. x1: coordenada x de la esquina superior izquierda.
4. y1: coordenada y de la esquina superior izquierda.
5. x2: coordenada x de la esquina inferior derecha.
6. y2: coordenada y de la esquina inferior derecha.

De la misma manera, el segundo nodo se encarga de cargar la red volumétrica y adquirir las detecciones del topic en el que la Yolov5 publica las imágenes directamente del topic original de imágenes. Cada vez que el nodo recibe información por ambos topics realiza una sincronización utilizando los timestamps presentes en las cabeceras de los mensajes. Con esto se asegura que las detecciones 2D corresponden correctamente con las imágenes que se están procesando. Una vez sincronizados los datos se ejecuta la red volumétrica, y siguendo la línea de la Yolov5, se publican los resultados es un nuevo topic */t4ac/perception/.../t4ac_3d_estimation_ros_node/3D_camera_obstacles/* en un mensaje específico para bounding boxes en 3D, con la siguiente estructura:

1. Type: un string que identifica que tipo de objeto es.
2. Score: un valor de 0 a 1 que identifica la confidencia de la detección.
3. Pose: centroide del objeto, conteniendo x, y, z y orientación.
4. L: longitud del objeto.
5. W: anchura del objeto.
6. H: altura del objeto.
7. Corners_3d: coordenadas de los vértices de la bounding box.
8. Vel_lin: velocidad lineal absoluta.
9. Vel_ang: velocidad angular absoluta.

Cabe destacar que al separar el sistema en dos nodos de ROS diferentes es posible utilizar las detecciones 2D en otras partes del proyecto sin necesidad de tener varios detectores bidimensionales funcionando al mismo tiempo.

7.2 Fusión sensorial y seguimiento

Una fusión sensorial es el proceso de combinación de datos y elementos procedentes de varios sensores diferentes con la finalidad de obtener resultados mas robustos que los que se obtienen en cada parte por separado. Hay varios tipos de fusión en función de en que parte de la línea de trabajo se realiza: late-fusión, middle-fusion y early-fusion.

A la par del trabajo desarrollado en este TFG, un compañero del grupo ha implementado un sistema de detección tridimensional basado en LiDAR [23]. Para compensar los puntos débiles de cada sensor se ha planteado realizar una late-fusion entre ambos sistemas. El procedimiento utilizado descrito en A.3 es un sencillo algoritmo que utiliza la distancia entre detecciones y el IoU3D para generar unas detecciones conjuntas mas robustas. En primer lugar calcula las distancias entre las detecciones de la red volumétrica que obtiene información de la cámara y las detecciones del sistema que utiliza LiDAR. Si la distancia entre detecciones es menor que cierto valor, se calcula el IoU3D para comprobar que las cajas intersectan y si es así se utiliza la bounding box proporcionada por el sistema basado en LiDAR.

Aplicando este sencillo algoritmo se consigue reducir en una gran medida el número de falsos positivos, ya que para que la fusión genere un objeto, este ha tenido que ser detectado por ambos sistemas. Cabe destacar que como bounding box 3d se utilizan las del LiDAR ya que son mucho mas precisas que las generadas por la red volumétrica.

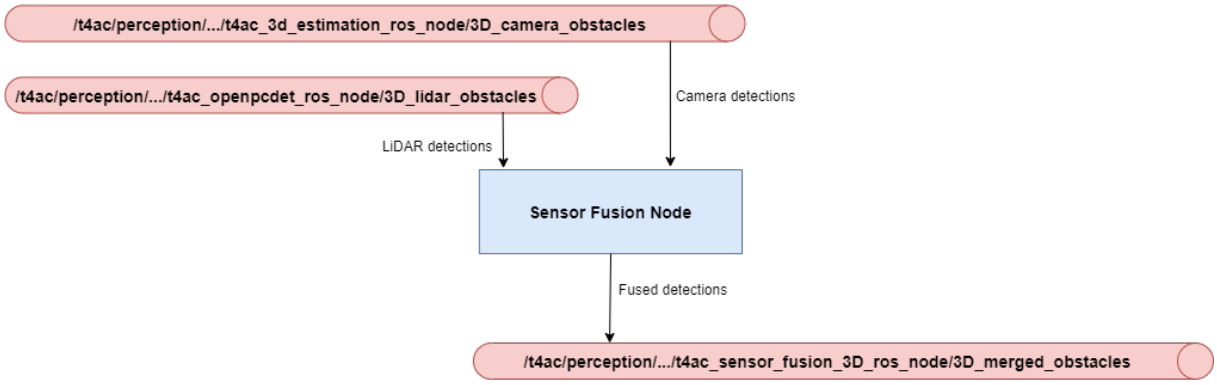


Figura 7.2: Esquema del funcionamiento del nodo de fusión sensorial

Esta fusión de información además de para mejorar resultados, se plantea como la entrada del sistema Smart-MOT [24] creado por compañeros del grupo. Smart-MOT es un sistema que realiza seguimiento en vista de pájaro a múltiples objetos de una escena.

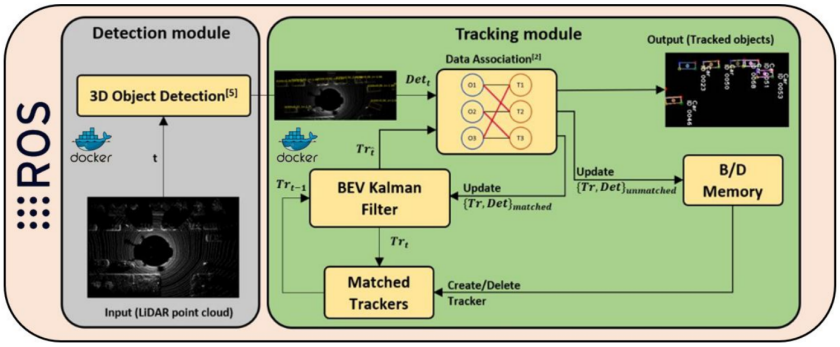


Figura 7.3: Esquema del funcionamiento de Smart-MOT

A partir de las detecciones tridimensionales se crea un mapa de la escena en vista de pájaro o BEV. Estas detecciones BEV se asocian con las trayectorias predichas para el mismo frame utilizando el algoritmo Húngaro. Una vez realizada la asociación se actualizan las trayectorias de ese frame. Por último las trayectorias y detecciones no asociadas se utilizan para crear o eliminar trayectorias que hayan desaparecido.

Capítulo 8

Resultados del sistema

Una vez implementado el sistema por completo y realizadas las correspondientes pruebas de funcionamiento, se ha procedido a obtener resultados sobre varias fuentes de datos para comprobar el funcionamiento real del sistema en múltiples ámbitos. Como se ha explicado en el capítulo 6, el sistema presenta errores si el objeto a estimar está a mucha distancia o muy cercano, es por esto que se aplica una etapa de filtrado y procesado. Todos los datos reflejados en este capítulo han pasado por este proceso.

8.1 Resultados sobre Kitti

En primer lugar se ha utilizado el dataset Kitti para obtener métricas y resultados. Como se ha explicado en el capítulo 5, para obtener las métricas oficiales se ha utilizado un código basado en el evaluador propio del dataset [22]. Dado que ninguna de las dos etapas que se utilizan se han entrenado sobre Kitti, se va a utilizar los datos de training como validación para obtener resultados extrapolables al benchmark sobre datos de test. Aplicando este código a los resultados obtenidos por la red volumétrica se obtienen los resultados de AP para detecciones y de AOS para orientación reflejados en la tabla 8.1.

Metric	Easy	Moderate	Hard
Car detection	0.8662	0.7086	0.5516
Car orientation	0.8650	0.7071	0.5498
Pedestrian detection	0.6519	0.5777	0.5052
Pedestrian orientation	0.6301	0.5538	0.4840

Tabla 8.1: Resultados del sistema sobre el evaluador de Kitti

Con estos resultados el sistema se encuentra en la posición 192 del benchmark oficial (ordenado por resultados de dificultad moderate), superando algunos detectores que se basan en información de LiDAR o realizan fusión de información como [25], que consigue un mAP del 68,78 %, pero quedando bastante por debajo de los sistemas con mejores resultados como [26] que llegan hasta el 82,54 %. Para complementar estos datos se han calculado además el IoU de la vista de pájaro (permite evaluar la intersección 3D sin tener en cuenta las alturas de los objetos) y la distancia media entre centroides (permite conocer el error medio en distancia de las detecciones) de nuevo sobre el dataset de Kitti, obteniendo un 35,48 % y 1,42m.

Figura 8.1: Detección de la red volumétrica sobre Kitti

Figura 8.2: Comparación de resultados de la red volumétrica (verde) con el ground truth de Kitti (rojo)

Se pueden observar mas imágenes en el siguiente enlace: https://youtu.be/2-wKPYg0xHs

8.2 Resultados cualitativos sobre Carla

Realizar una evaluación sobre un simulador como Carla permite comprobar como se desenvuelve la red volumétrica en un entorno dinámico antes de comprobar su funcionamiento en real. Cabe destacar que los datos del entorno que se utilizan proceden de Rosbags grabados sobre simulación.

Un caso de uso es una sucesión de eventos que ocurren en simulación o real, y que suele representar situaciones habituales en conducción autónoma. En lugar de recorrer la simulación libremente, se van a utilizar ciertos casos de uso para comprobar como responde el sistema ante diferentes situaciones, así como poder comparar los resultados con otros detectores.

Uno de los que se ha utilizado es *Pedestrian Crossing*, en el que el vehículo se aproxima a un paso de peatones en el que hay una persona cruzando.

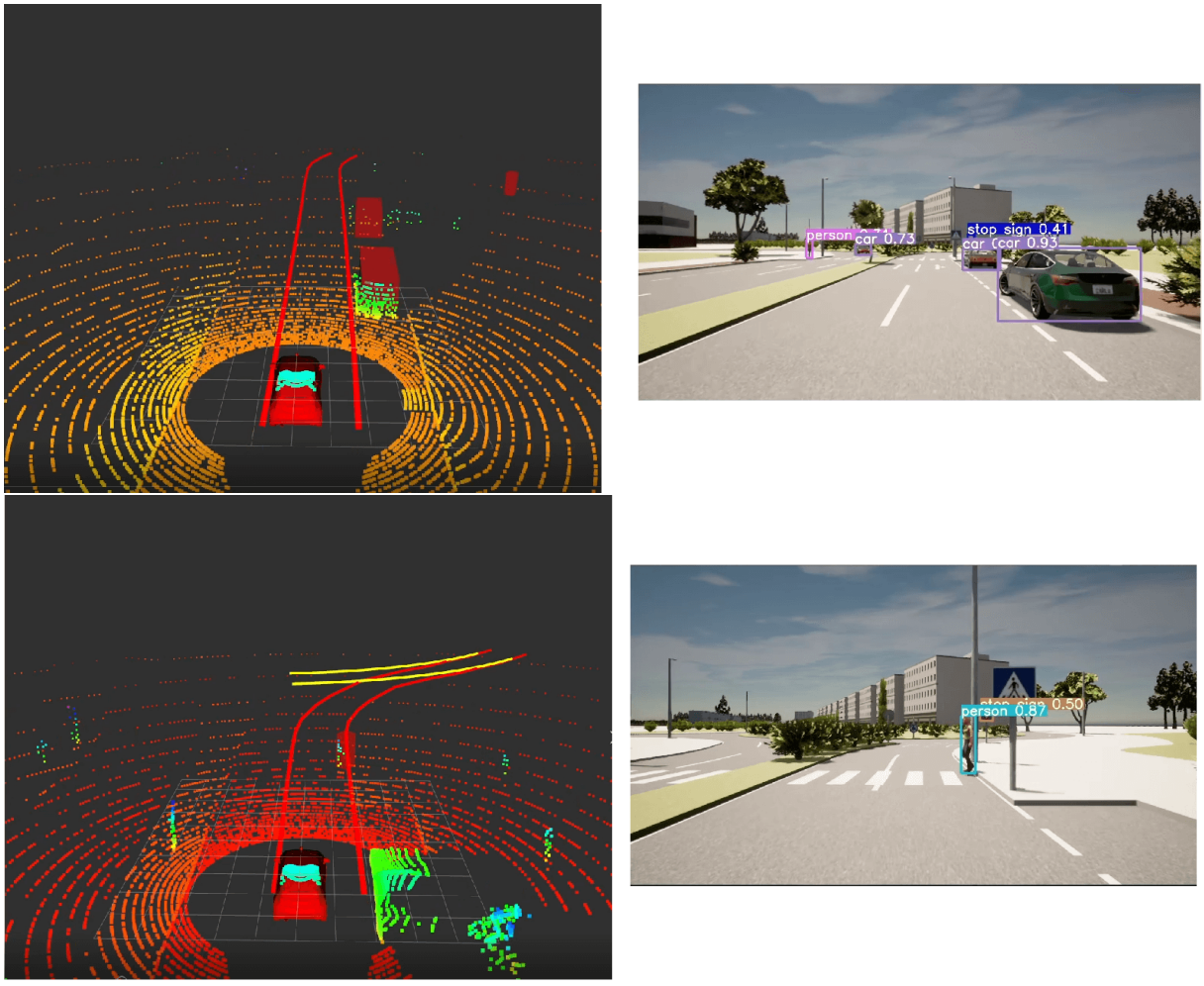


Figura 8.3: Detección de la red volumétrica (cubos rojos) y de la Yolov5 (a la derecha) sobre el caso simulado *Pedestrian Crossing*

En la figura 8.3 se observa como al aproximarse al paso el sistema localiza los coches aparcados en el margen derecho, si las detecciones se acercan al vehículo en movimiento el filtro actúa y las elimina para evitar datos erróneos. Una vez detenidos el sistema detecta de una manera bastante robusta al peatón mientras cruza la calzada.

Otro de los escenarios sobre los que se han hecho pruebas es *Give Way*, en el que el vehículo llega a una rotonda y cede el paso a otro coche que ya se encuentra en ella.

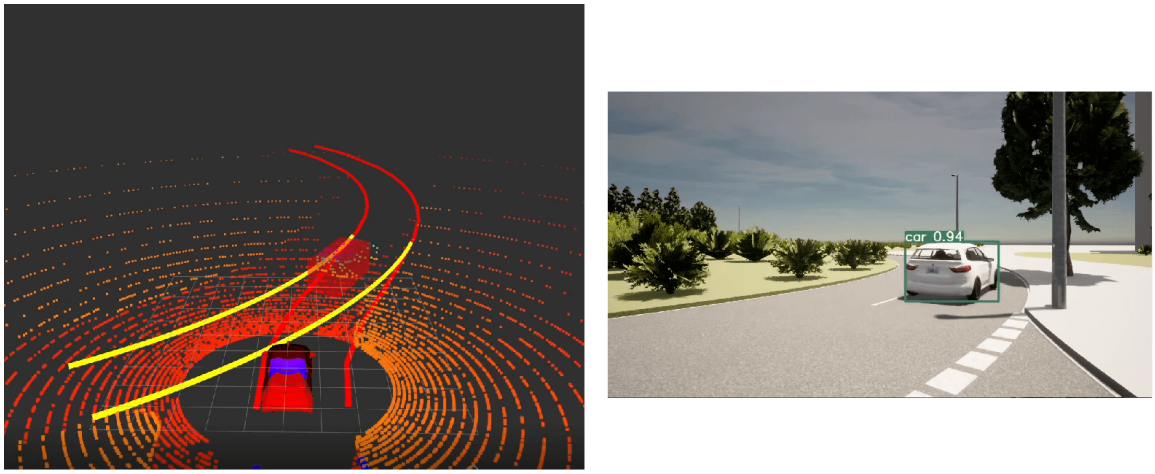


Figura 8.4: Detección de la red volumétrica (cubos rojos) y de la Yolov5 (a la derecha) sobre el caso simulado *Give Way*

En los resultados de la figura 8.4 se observa que tanto la Yolov5 como la red volumétrica detectan correctamente al vehículo que está en el interior de la rotonda.

La sensación general obtenida es que la red volumétrica proporciona unas detecciones lo bastante robustas como para poderse utilizar en otras tareas o realizar fusión con las detecciones de otros sensores, tal y como se ha comentado en el apartado 7.2. Por otra parte, ocasionalmente las bounding box tridimensionales localizan correctamente al objeto pero sufren rotaciones o traslaciones debido a los errores introducidos por la perspectiva del objeto en los cálculos utilizando geometría y proyecciones. Estos errores que se vuelven mayores cuanto mas lejos se encuentre el objeto o en el caso concreto de encontrarse en el borde del campo de visión de la cámara.

Ambos casos de uso se pueden visualizar en el siguiente enlace: https://youtu.be/A_BrOR7PJRQ

8.3 Resultados cualitativos sobre el vehículo real

Por último se ha comprobado el funcionamiento sobre el vehículo autónomo del grupo. Al igual que en el caso de simulación en Carla, se han grabado una serie de Rosbags de casos de uso, esta vez en un tramo del campus externo de la UAH.

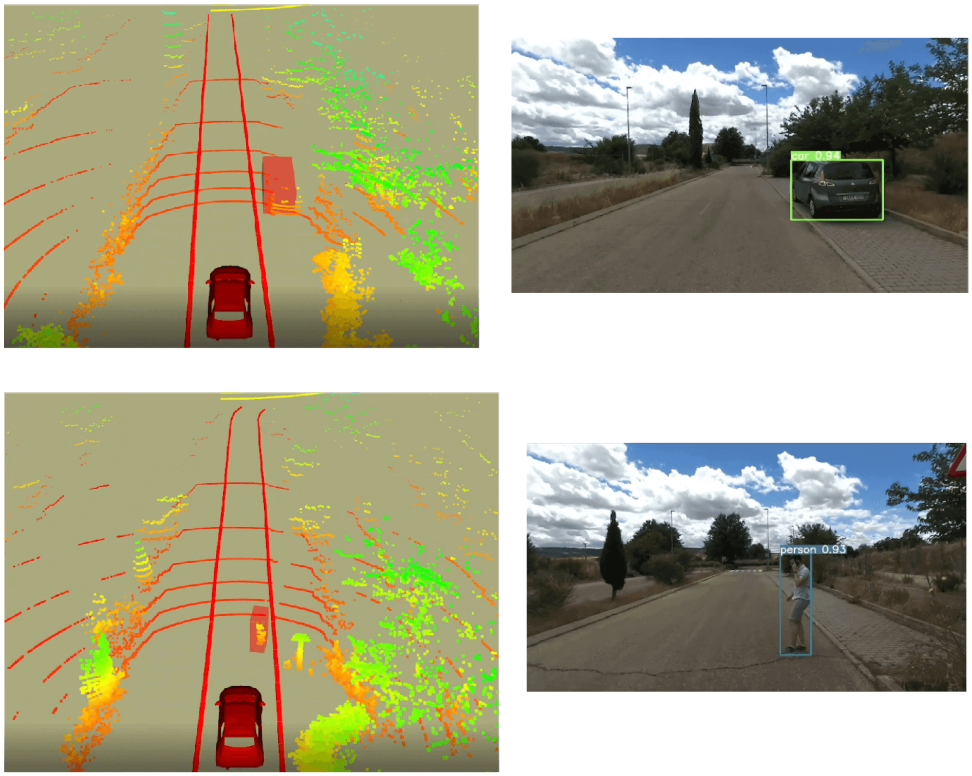


Figura 8.5: Detección de la red volumétrica sobre el caso real *unexpected*

Este caso es similar al *Pedestrian Crossing* del apartado 8.2 solo que el peatón aparece de forma abrupta en medio de la calzada en lugar de por un paso de peatones. Se observa como el sistema realiza detecciones bastante robustas también en real, pero sigue presentando los mismos problemas de offset y rotaciones que en simulación.

Se puede visualizar el caso completo en https://youtu.be/N-XFtglpIQc

8.4 Resultados cualitativos de la fusión sobre el vehículo real

De la misma manera que para la red volumétrica, se han utilizado varios Rosbags de casos de uso sobre el vehículo real para obtener resultados cualitativos. Se han utilizado los casos *Unexpected*, en el que el vehículo se detiene para permitir el paso a un peatón, y el caso *Give Way*, en el que el vehículo cede el paso en una rotonda.

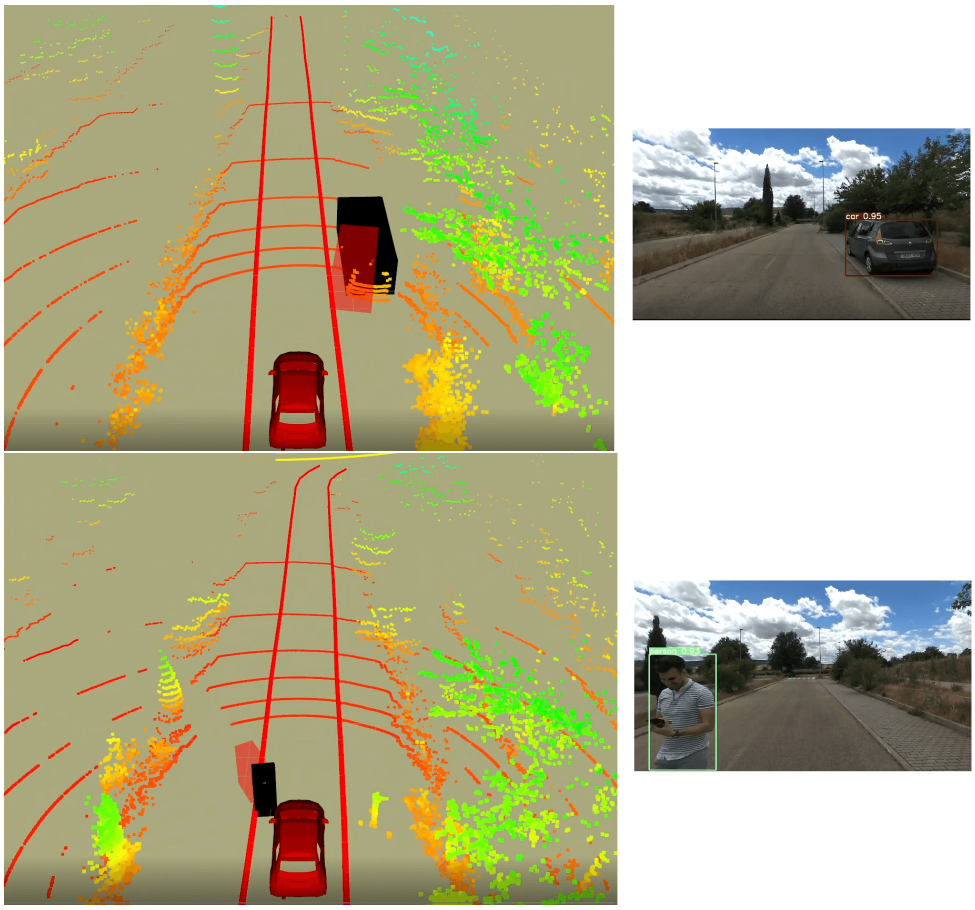


Figura 8.6: Resultados de la fusión (negro) y de la red volumétrica (rojo) en real sobre el caso Unexpected

Figura 8.7: Resultados de la fusión (negro) y de la red volumétrica (rojo) en real sobre el caso Give Way

En las figuras 8.6 y 8.7 se han reflejado instantes en los que la red volumétrica genera resultados con errores que son corregidos fácilmente gracias a la late-fusion con el LiDAR, mejora que se puede apreciar en https://youtu.be/yj4lTBX6ggg.

Capítulo 9

Evaluador propio

9.1 Idea de evaluador

Un benchmark de evaluación es una herramienta que permite calcular varias métricas para medir el rendimiento de sistemas basados en Deep Learning de una forma equitativa. En el caso de sistemas de detección, permite medir cuanto se acercan los resultados proporcionados por el detector a los elementos que se están detectando.

En este trabajo se ha realizado un desarrollo conjunto con J. de la Peña [23] para integrar un sistema de evaluación de la capa de percepción en el Autonomous Driving Benchmark Development Kit que está siendo desarrollado en conjunto por Universidad de Alcalá y Karlsruhe Institute of Technology.

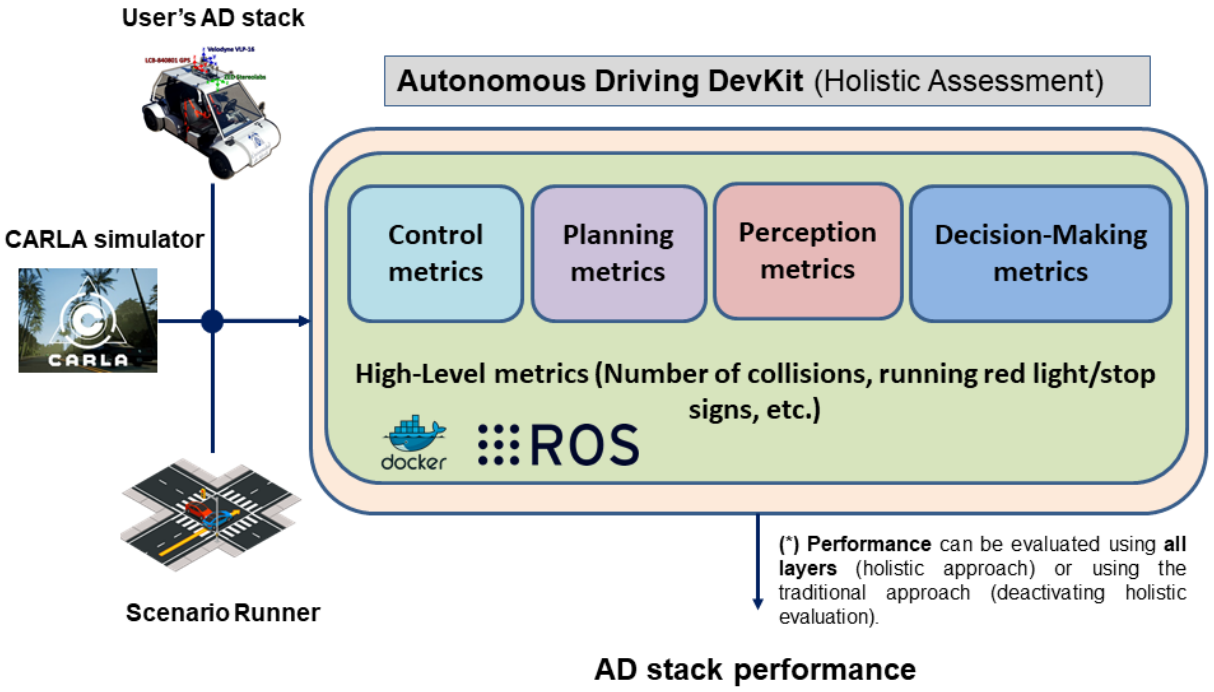


Figura 9.1: Esquema del Ad Devkit

Se plantea utilizar el simulador Carla para generar el ground-truth utilizando de nuevo nodos de ROS, y a partir de estos datos ejecutar el sistema que se quiere evaluar y generar las métricas con los resultados.

9.2 Métricas generadas

Para realizar la evaluación se generan una serie de métricas comparando los resultados del sistema con el ground truth generado por nuestro evaluador.

1. IoU: una de las métricas mas utilizadas en detección es Intersection over Union. Se puede definir en 2D haciendo la relación entre el área de intersección de dos bounding box y la unión, o en 3D si se hace con volúmenes. Permite conocer lo similares que son dos bounding box.

$$IoU(2D) = \frac{\text{Área de intersección}}{\text{Área de unión}}$$
$$IoU(3D) = \frac{\text{Volúmen de intersección}}{\text{Volúmen de unión}}$$

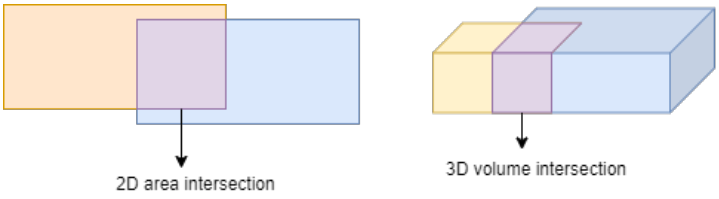


Figura 9.2: Representación métrica IoU del evaluador

   El evaluador desarrollado realiza el cálculo de ambos tipos de IoU, sin embargo, es el valor tridimensional el que se devuelve al usuario.

2. Precision: capacidad del modelo de detectar los elementos para los que ha sido entrenado.

$$Precision = \frac{\text{True Positives}}{\text{True positives + False positives}}$$

3. Recall: capacidad del modelo para detectar todos los objetos del ground truth.

$$Recall = \frac{\text{True Positives}}{\text{True positives + False negatives}}$$

4. Curva Precision-Recall: representa en la misma gráfica los diferentes valores de precision y recall que se obtienen utilizando diferentes umbrales de score, observando el rendimiento del sistema a evaluar.

5. Average Precision y mAP: se puede resumir la información de la curva precision-recall calculando el área bajo la curva, dando lugar al AP. Esta métrica permite comparar dos sistemas de manera mas sencilla y sin necesidad de utilizar gráficas. Si además realizamos la media entre el AP de las diferentes clases se obtiene el mAP.

6. mAVE: en caso de que el sistema a evaluar proporcione información de las velocidades de los objetos, se calcula el error de velocidad, calculando el módulo del vector diferencia entre los vectores velocidad del ground truth y de la detección.

9.3 Implementación

El evaluador cuenta con un generador de ground truth sobre el simulador Carla implementado sobre nodos de ROS. Al ejecutar simulaciones o Rosbags el nodo generador se encarga de adquirir la información de los diferentes sensores (cámara, LiDAR ...) y elementos de la simulación (vehículos, peatones ...) para generar los datos necesarios. La información procesada se almacena en un archivo csv con el siguiente formato:

Nombre	Descripción
Frame	Identificador del frame de simulación al que pertenecen los datos
Timestamp	Valor de tiempo en segundos asociado al frame
Id	Identificador en simulación del objeto
Type	Clase a la que pertenece el objeto
Alpha	Ángulo de observación del objeto (equivalente a Kitti)
Left, Top	Coordenadas píxel del punto superior izquierdo de la bounding box 2D sobre la cámara.
Right, Bottom	Coordenadas píxel del punto inferior derecho de la bounding box 2D sobre la cámara.
H, W, L	Dimensiones del objeto.
X, Y, Z	Coordenadas del centroide del objeto
Rotation Z	Rotación del objeto sobre el eje Z (vertical)
vx, vy, vz	Componentes de velocidad de un objeto sobre cada eje.

Tabla 9.1: Elementos del archivo csv del evaluador

Además del archivo csv, el nodo generador se encarga de guardar las imágenes de la cámara, la nube de puntos de LiDAR y la información del radar en sus respectivos archivos para poder realizar la detección sobre ellos.

Una vez generados los archivos se debe ejecutar el sistema que se quiera evaluar y guardar los resultados en otro archivo csv que siga la misma estructura que el ground truth. Con ambos csv ya se puede ejecutar la segunda parte del evaluador, que se encarga de cargar los datos de ambos archivos y generar las métricas correspondientes.

En esta primera versión del evaluador se realizan las asociaciones de detecciones y ground truth utilizando el IoU3D y la distancia entre los centroides. Para que una detección pueda ser asociada con cierto ground truth debe de encontrarse a menos de 2 metros y tener un IoU3D superior al 25 %. De todas las que cumplan con estas condiciones se selecciona la que presente menos distancia con el ground truth.

Cabe destacar que han surgido varios problemas relacionados con el simulador y ROS. En primer lugar, en la versión sobre la que se trabaja, Carla solo identifica dos clases de elementos, Car y Pedestrian, lo que imposibilita evaluar el resto. Por otra parte a la hora de utilizar las imágenes de la cámara surgen problemas de sincronización con el resto de topics, lo que puede empeorar considerablemente el resultado, por lo que se decide no aplicar el evaluador a la red volumétrica. Estos problemas se plantean abordar en trabajos futuros.

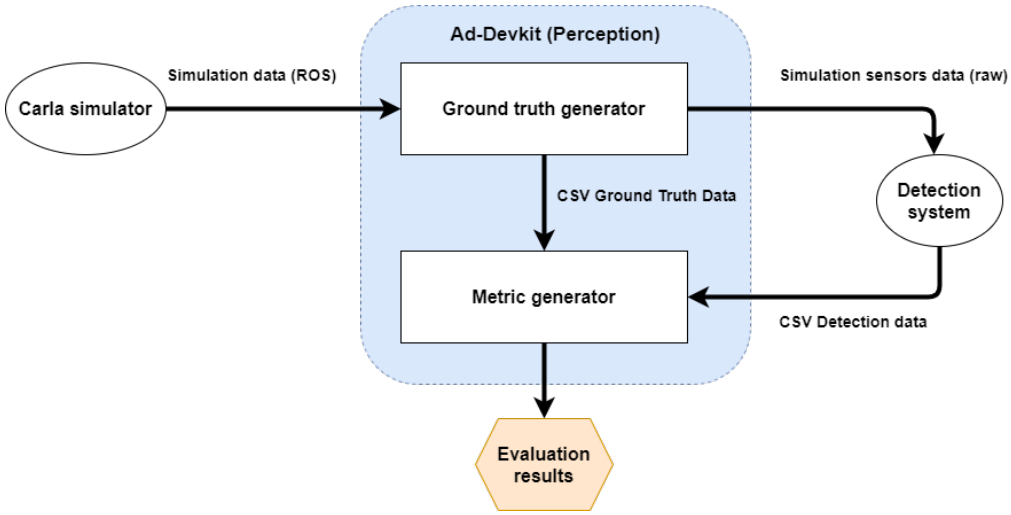


Figura 9.3: Esquema del funcionamiento del evaluador de percepción

9.4 Resultados del evaluador

Para comprobar el funcionamiento del sistema de evaluación implementado, se ha evaluado el sistema de detección basado en LiDAR de Javier de la Peña [23] sobre el escenario *Give Way* utilizado en el apartado 8.2, en el que el vehículo se aproxima a una rotonda y debe ceder el paso a otro que ya se encuentra en su interior. Se han obteniendo los siguientes datos:

Métrica	Valor
AP (Car)	35 %
mIoU (Car)	49.26 %
mAVE (Car)	1.48 m/s

Tabla 9.2: Resultados del evaluador sobre PointPillars Multihead en Give Way en simulación para la clase Car

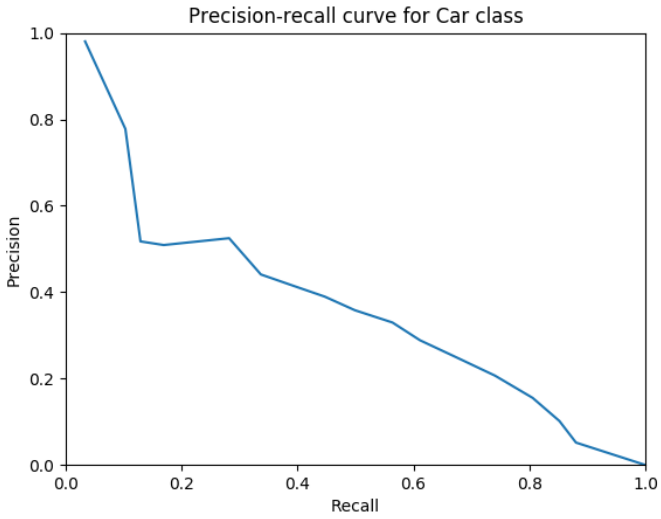


Figura 9.4: Precision-recall de PointPillars Multihead sobre Give Way en simulación para la clase Car

Se observa que al aplicar el sistema sobre simulación penaliza considerablemente los resultados, especialmente por la inexistencia de intesidad en los datos de LiDAR y por la inestabilidad que sufren los Rosbags en algunas situaciones.

Capítulo 10

Conclusiones y trabajos futuros

En el trabajo realizado estos meses se ha conseguido implementar un sistema de detección 3D a partir únicamente de la información 2D de la cámara. Si bien presenta resultados bastante fiables, se plantean una serie de líneas de trabajo a seguir para lograr robustecer los resultados:

1. Ya que el vehículo cuenta con una cámara estéreo, se pretende utilizar esta información de profundidad para mejorar el proceso de detección.
2. Realizar modificaciones sobre la red volumétrica y entrenar de nuevo.
3. Terminar de implementar el tracking utilizando Smart-MOT o estudiar alguna otra técnica.

En los sucesivos años se pretende continuar el trabajo de investigación realizado durante los últimos meses en el grupo RobeSafe para conseguir un sistema de detección mas robusto en el proyecto. Las líneas generales que se plantean continuar son las siguientes:

1. Se pretende mejorar el algoritmo utilizado para fusión sensorial entre diferentes sensores. Se puede plantear además técnicas de early o middle fusion.
2. Buscar otros métodos para realizar la detección 3D.
3. Continuar con el desarrollo del Ad-Devkit.

Bibliografía

[1] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16. [Online]. Available: https://carla.org/

[2] "Opendrive road standard." [Online]. Available: https://www.asam.net/standards/detail/opendrive/

[3] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," 2019.

[4] R. Girshick, "Fast r-cnn," 2015.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.

[6] M. Takahashi, A. Moro, Y. Ji, and K. Umeda, "Expandable yolo: 3d object detection from rgb-d images," 2020.

[7] "Universidad de chuo, japón." [Online]. Available: http://global.chuo-u.ac.jp/english/

[8] H. Königshof, N. O. Salscheider, and C. Stiller, "Realtime 3d object detection for automated driving using stereo vision and semantic information," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 1405–1410.

[9] B. Ranft and T. Strauß, "Modeling arbitrarily oriented slanted planes for efficient stereo vision based on block matching," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014, pp. 1941–1947.

[10] N. O. Salscheider, "Simultaneous object detection and semantic segmentation," 2020.

[11] Z. Wu, C. Shen, and A. van den Hengel, "Wider or deeper: Revisiting the resnet model for visual recognition," 2016.

[12] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," 2016.

[13] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[14] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," 2017.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[16] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[17] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[18] G. Jocher, "Yolov5." [Online]. Available: https://github.com/ultralytics/yolov5

[19] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," 2020.

[20] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[21] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[22] C. Guindel, "A evaluation code based on kitti object development kit." [Online]. Available: https://github.com/cguindel/eval_kitti

[23] J. de la Peña, "Algoritmos de detección de objetos 3d basados en lidar: comparación entre técnicas clásicas y deep learning," Trabajo de Fin de Grado, Universidad de Alcalá, Escuela Politécnica Superior, 2021.

[24] C. Gómez-Huélamo, J. Del Egido, L. M. Bergasa, R. Barea, M. Ocaña, F. Arango, and R. Gutiérrez-Moreno, "Real-time bird's eye view multi-object tracking system based on fast encoders for object detection," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, pp. 1–6.

[25] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3d object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[26] W. Zheng, W. Tang, L. Jiang, and C.-W. Fu, "Se-ssd: Self-ensembling single-stage object detector from point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 14 494–14 503.

Apéndice A

Algoritmos

A.1 Non Max Suppression

Data: D = lista de detecciones ordenadas por score descendente
Result: OUT = detecciones filtradas
begin
    OUT = lista vacía;
    for *detección in D* **do**
        Añadir detección a la lista OUT;
        Borrar detección de la lista D;
        for *elemento in D* **do**
            Calcular el IoU de detección con elemento;
            if $IoU > threshold$ **then**
                Borrar elemento de D;

Algoritmo A.1: Algoritmo Non Max Suppression

A.2 Selective search algorithm used in RCNN

Data: Input image
Result: Region proposals (aproximadamente 2000)
begin
    Generar la mayor cantidad de regiones utilizando una segmentación inicial;
    Combinar regiones similares para formar elementos mas grandes ;
    Generar las region proposals a partir de las regiones anteriores;

Algoritmo A.2: Algoritmo de búsqueda selectiva para RCNN

A.3 Algoritmo de late sensor fusion

Data: 3D camera detections;
3D LiDAR detections;
Result: 3D fusion detections
begin
    Calcular para cada detección de cámara la distancia a cada detección de LiDAR;
    if *La distancia entre dos detecciones es menor a un valor límite (3m)* **then**
        Se calcula el IoU3D entre ambas cajas;
        if *El IoU3D es mayor a 0* **then**
            Se utiliza la caja procedente del LiDAR como bounding box del objeto

Algoritmo A.3: Algoritmo de late sensor fusion

Apéndice B

Pliego de condiciones

Las herramientas necesarias para la elaboración del proyecto han sido:

- PC Ryzen 7 3700X y Nvidia RTX 2060 Super
- Sistema operativo Ubuntu 18
- Visual Studio Code
- Python
  - PyTorch 1.7.1
  - Torchvision 0.8.2
  - Numpy 1.20.0
  - OpenCV 4.2.0.32
  - Pandas 1.2.2
  - Matplotlib 3.3.4
  - Math
- Carla Simulator (sección 2.1)
- Docker (sección 2.3)
- ROS (Robot Operative System)
- GitHub (https://github.com/RobeSafe-UAH/t4ac_3d_estimation_ros)
- Kitti dataset (sección 5)
- LaTeX y TexMaker

Apéndice C

Presupuesto

En este apartado se detallan los costes para la realización de este trabajo, actualizados a fecha de agosto del 2021. Se deben tener en cuenta los costos de materiales, tanto software como hardware, así como las horas de trabajo dedicadas.

El desarrollo se ha realizado sobre el hardware especificado en la tabla C.1.

Nombre	Precio
NVIDIA RTX 2060 Super	410 €
AMD Ryzen 7 3700X	360 €
Corsair Vengance 2x16 GB	168 €
Corsair 650W 80 Plus Gold (TX650M)	71 €
Toshiba TR200 SSD 480 GB	44 €
TOTAL	1053 €

Tabla C.1: Elementos hardware principales del ordenador utilizado

El software utilizado no conlleva coste ya que se utilizan licencias de código abierto o libres, tanto para el desarrollo (CARLA, ROS, Linux ...) como para la escritura (LaTex). Se debe tener en cuenta el coste del propio trabajo realizado. Se aplica un salario de 15 € por hora trabajada con una jornada de media de 3,5 horas.

Tarea	Días aproximados	Coste
Estudio del estado del arte y de herramientas software	30	1575 €
Implementación inicial del sistema (Yolov5 y volumétrica)	20	1050 €
Generación de resultados y pruebas sobre Kitti	10	525 €
Adaptación del sistema al proyecto T4AC sobre ROS	20	1575 €
Implementación del evaluador sobre Carla (Ad-Devkit)	30	1575 €
Generación de resultados sobre Rosbags de simulación y reales	10	525 €
Escritura de la memoria	45	2362 €
TOTAL	165	9187 €

Tabla C.2: Horas de trabajo realizado

Sumando todos los costes se obtiene un valor de 10240 €, al que se le aplica el 21 % de IVA estipulado para llegar a la cifra final de **12390** €.

Apéndice D

Manual de usuario

D.1 Configuración inicial

Para configurar la red volumétrica en el Docker del proyecto es necesario descargar los pesos y el nodo de ROS:

```
cd /home/robesafe/models/
mkdir 3d_estimation && cd 3d_estimation && mkdir weights_3D_est
cd weights_3D_est
wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=
    $(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-
    certificate 'https://docs.google.com/uc?export=download&id=1yEiquJg9inIFgR3F-
    N5Z3DbFnXJ0aXmA' -O- | sed -rn 's/.*confirm=([0-9A-Za-z_]+).*/\1\n/p')&id=1
    yEiquJg9inIFgR3F-N5Z3DbFnXJ0aXmA" -O epoch_10.pkl && rm -rf /tmp/cookies.txt

cd /home/robesafe/t4ac_ws/src/t4ac_architecture/t4ac_perception_layer/detection/camera
git clone https://github.com/RobeSafe-UAH/t4ac_3d_estimation_ros
```

Es posible que en versiones futuras de la imagen algún paso no sea necesario o haya variado.

D.2 Ejecución del proyecto junto a la red volumétrica

Para la ejecución se requieren los siguientes alias en bashrc:

```
export CARLA_ROOT=~/carla/
alias World="cd ${CARLA_ROOT}/Dist/CARLA_Shipping_0.9.9.4-113-gc18ccca7-dirty/
    LinuxNoEditor && ./CarlaUE4.sh -world-port=2000 -resx=800 -resy=600"
alias Bridge="roslaunch carla_ros_bridge carla_ros_bridge_with_example_ego_vehicle.launch"
alias T4AC_Project="roslaunch t4ac_utils t4ac_config.launch"
```

Para configurar las capas y paquetes que se desean ejecutar se debe editar el archivo */home/robesafe/t4ac_ws/src/t4ac_architecture/t4ac_utils/launch/t4ac_config.launch*

D.2.1 Ejecución sobre simulación

```
World
Bridge
T4AC_Project simulation:=true lidar_camera_fusion:=false
```

Utilizando la ventana del Bridge podemos desplazarnos por la simulación y con Rviz observar los diferentes topics.

D.2.2 Ejecución sobre Rosbags

Importante especificar en el comando de ejecución si el Rosbag se trata de una simulación o del vehículo real y el directorio del archivo .bag que se quiere reproducir.

```
T4AC_Project simulation:=true lidar_camera_fusion:=false
rosbag play /.../####.bag
```

De nuevo se utiliza Rviz para visualizar los diferentes topics.

D.3 Ejecución del Ad-Devkit

En primer lugar se debe modificar la ruta del rosbag a evaluar en el archivo */home/robesafe/t4ac_ws/src/t4ac_carla_simulator/ad_devkit/launch/ad_devkit.launch*.

Generación del ground truth en */home/robesafe/t4ac_ws/src/t4ac_carla_simulator/ad_devkit/ databases/perception*:

```
roscore &
roslaunch ad_devkit ad_devkit.launch
```

Sobre los datos generados se ejecuta el sistema a evaluar y se genera un csv con las detecciones.

Este archivo se debe guardar en */home/robesafe/t4ac_ws/src/t4ac_carla_simulator/ad_devkit/ perception/camera* en caso de ser cámara o */home/robesafe/t4ac_ws/src/t4ac_carla_simulator/ ad_devkit/databases/perception/lidar* en caso de ser LiDAR, con el nombre *detections.csv*.

Para obtener las métricas:

```
cd /home/robesafe/t4ac_ws/src/t4ac_carla_simulator/ad_devkit/src/perception/evaluator
python evaluate.py
```

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá